

# Systematische Prozessunterstützung für die Entwicklung laufzeitkritischer Softwaresysteme

---

## PROKRIS-Methodik und -Framework

### **Dissertation**

zur Erlangung des akademischen Grades  
Doktoringenieur (Dr.-Ing.)

vorgelegt an der  
Technischen Universität Dresden  
Fakultät Informatik

eingereicht von

**Dipl.-Ing. Simone Röttger**  
**(Diplomsoftwaretechnologin)**  
geboren am 24.07.1970 in Zittau

#### **Gutachter:**

Prof. Dr. rer. nat. habil. Heinrich Hußmann (Ludwig-Maximilians-Universität München)

Prof. Dr. rer. nat. habil. Uwe Aßmann (Technische Universität Dresden)

#### **Tag der Verteidigung:**

Dresden, den 16. Oktober 2009



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Problemstellung . . . . .	2
1.2	Zielstellung . . . . .	3
1.3	Ergebnisse . . . . .	5
1.4	Aufbau der Arbeit . . . . .	7
<b>2</b>	<b>Grundlagen</b>	<b>9</b>
2.1	Der Begriff „nicht-funktionale Eigenschaft (NFE)“ . . . . .	9
2.2	Klassifikation nicht-funktionaler Eigenschaften . . . . .	13
2.3	Grundbegriffe von Entwicklungsprozessen . . . . .	17
2.4	Prozessbewertung . . . . .	24
2.5	Bekannte Vorgehens- und Prozessmodelle . . . . .	27
2.6	Modellgetriebene Softwareentwicklung . . . . .	31
2.7	Wiederverwendung von Vorgehens- und Prozessspezifikationen . . . . .	33
2.8	Beschreibungstechniken für Prozesse . . . . .	35
2.9	Ausführung von Prozessmodellen . . . . .	41
<b>3</b>	<b>Die Idee des PROKRIS-Ansatzes</b>	<b>45</b>
3.1	Die Herausforderungen laufzeitkritischer NFE an Vorgehensmodelle . . . . .	45
3.2	Anforderungen an die systematische Prozessunterstützung der Entwicklung laufzeitkritischer Systeme . . . . .	54
3.3	Der PROKRIS-Ansatz als Lösung . . . . .	55
3.4	Einsatzszenarien und Rollen . . . . .	58
3.5	Zusammenfassung . . . . .	60
<b>4</b>	<b>Erweitertes Vorgehensmodell zur Entwicklung laufzeitkritischer Software</b>	<b>61</b>
4.1	NFE-Unterstützung in bestehenden Prozessmodellen . . . . .	61
4.2	Notwendige Eigenschaften des erweiterten Vorgehensmodells . . . . .	71
4.3	NFE-Prozessmuster für laufzeitkritische NFE . . . . .	78
4.4	Zusammenfassung . . . . .	92
<b>5</b>	<b>PROKRIS-Methodik: Kontextbasierte Anpassung der erweiterten Vorgehensmodelle für laufzeitkritische Software</b>	<b>95</b>
5.1	Konzepte der PROKRIS-Methodik . . . . .	95
5.2	Modellierungsdimensionen . . . . .	97
5.3	Einheitliche Beschreibungstechnik . . . . .	102
5.4	Wiederverwendung . . . . .	102

5.5	Kontexte und Facettenklassifikation . . . . .	107
5.6	Prozesskomposition . . . . .	112
5.7	Anwendung der PROKRIS-Methodik zur Komposition und Anpassung der erweiterten Vorgehensmodelle . . . . .	114
5.8	Ausführungsunterstützung innerhalb eines Projekts . . . . .	122
5.9	Subprozesse - Parallelität in der Entwicklung . . . . .	125
5.10	Zusammenfassung . . . . .	128
<b>6</b>	<b>Die PROKRIS-Prozessbibliothek</b>	<b>129</b>
6.1	Verwendete Prozessbeschreibungsmittel . . . . .	129
6.2	Erweiterungen des Prozessmetamodells SPEM 2.0 . . . . .	130
6.3	Umsetzung des Metamodells in Werkzeugunterstützung . . . . .	139
6.4	Zusammenfassung . . . . .	142
<b>7</b>	<b>Der PROKRIS-Prozesskomposer</b>	<b>143</b>
7.1	Erweiterte Metamodellkonzepte zur Komposition . . . . .	143
7.2	Tailoring und Konkretisierung der statischen Modelle . . . . .	144
7.3	Übersetzung der statischen in ausführbare Vorgehensmodelle . . . . .	146
7.4	Individualisierung . . . . .	147
7.5	Zusammenfassung . . . . .	148
<b>8</b>	<b>Ausführungsumgebung: Erweiterte Workflowumgebung für PROKRIS- Pro- zessmodelle</b>	<b>149</b>
8.1	Erweiterung der Standardarchitektur der WfMC . . . . .	149
8.2	Die Architektur der PROKRIS-Ausführungsumgebung . . . . .	153
8.3	Details der prozessbasierten Ausführungsunterstützung . . . . .	155
8.4	Zusammenfassung . . . . .	160
<b>9</b>	<b>Evaluierung von PROKRIS-Methodik und -Framework</b>	<b>163</b>
9.1	Gewährleistung der Eigenschaften von Vorgehensmodellen zur Entwick- lung laufzeitkritischer Software . . . . .	163
9.2	Fallstudien zu NFE-spezifischen Vorgehensmodellen . . . . .	165
9.3	Kombination mit anderen Softwareentwicklungsprozessen . . . . .	166
9.4	Beispielanwendung und Werkzeugunterstützung . . . . .	168
9.5	Zusammenfassende Bewertung der methodischen und technischen Konzepte	177
<b>10</b>	<b>Verwandte Forschungsarbeiten</b>	<b>179</b>
<b>11</b>	<b>Zusammenfassung und Ausblick</b>	<b>185</b>
11.1	Zusammenfassung . . . . .	185
11.2	Wesentliche Ergebnisse der Arbeit . . . . .	187
11.3	Geltungsbereich des Ansatzes . . . . .	189
11.4	Weiterführende Arbeiten . . . . .	190
<b>A</b>	<b>Katalog der Prozessmuster</b>	<b>193</b>

A.1	Explizite Spezifikation der NFE . . . . .	194
A.2	Identifikation der kritischen Szenarien . . . . .	196
A.3	Konfliktanalyse . . . . .	198
A.4	Verifikationsschleife . . . . .	200
A.5	Additive Verifikation der NFE . . . . .	201
A.6	Transformation der NFE-Spezifikationen . . . . .	203
A.7	Analyse der Operationalisierung . . . . .	205
A.8	Dekomposition der NFE . . . . .	206
A.9	Komposition der NFE . . . . .	208
A.10	Entwicklung entlang des kritischen Pfades . . . . .	210
A.11	Inkrementelle Entwicklung der kritischen Systembereiche . . . . .	211
A.12	NFE-spezifische Sprache . . . . .	213
A.13	NFE-spezifische Abbildungsbeschreibung . . . . .	215
<b>B</b>	<b>Rollen und deren Aufgaben</b>	<b>217</b>
B.1	Anwendungsentwicklung . . . . .	217
B.2	NFE-Domäneneentwicklung . . . . .	218
<b>C</b>	<b>Ressourcenspezifikation</b>	<b>221</b>
<b>D</b>	<b>Vorgehensmodellspezifikationen der Fallstudienbeispiele</b>	<b>225</b>
D.1	Erweitertes abstraktes NFE-Vorgehensmodell . . . . .	225
D.2	Realzeit-spezifisches Vorgehensmodell . . . . .	226
D.3	Sicherheits-spezifisches Vorgehensmodell . . . . .	227
D.4	Ausführungsspezifikation des realzeit-spezifischen Vorgehensmodells . . . .	227
	<b>Literaturverzeichnis</b>	<b>231</b>



# Abkürzungsverzeichnis

<b>aPDL</b>	Abstract Process Definition Language
<b>CIM</b>	Computation Independent Model
<b>CMMI</b>	Capability Maturity Model Integration
<b>CQML</b>	Component QoS Modeling Language
<b>EPF</b>	Eclipse Process Framework
<b>JBPM</b>	Java Business Process Management
<b>jPDL</b>	Java Process Definition Language
<b>MDA</b>	Model Driven Architecture
<b>MDSD</b>	Model Driven Software Development
<b>MDV</b>	Model Driven Verification
<b>MOF</b>	Meta Object Facility
<b>NFE</b>	Nicht-funktionale Eigenschaft
<b>NFR</b>	Non-Functional Requirement
<b>OCL</b>	Object Constraint Language
<b>OMG</b>	Object Management Group
<b>PIM</b>	Platform Independent Model
<b>PMI</b>	Project Management Institute
<b>PROKRIS</b>	Prozessunterstützung für die Entwicklung lauffzeitkritischer Softwaresysteme
<b>PSM</b>	Platform Specific Model
<b>QS</b>	Qualitätssicherung
<b>RT</b>	Real-Time
<b>RUP</b>	Rational Unified Process
<b>SPE</b>	Software Performance Engineering
<b>UML</b>	Unified Modeling Language
<b>XML</b>	Extended Markup Language
<b>XP</b>	Extreme Programming
<b>WfMC</b>	Workflow Management Coalition
<b>WCET</b>	Worst Case Execution Time





# 1 Einleitung

Laufzeitkritische Systeme werden in unserer technischen Umgebung nicht nur immer verbreiteter, auch deren Auswirkungen für Leib und Leben sowie wirtschaftliche Werte im Versagensfall werden immer kritischer. Beispiele hierfür sind in vielen Bereichen des täglichen Lebens zu finden, angefangen von der Steuerung im Auto, die zunehmend durch komplexe Software erfolgt, über sicherheitskritische Internetanwendungen, wie Online-Banking, bis hin zur Steuerungssoftware in Flugzeugen oder medizinisch-technischen Instrumenten. Bei der Entwicklung derartiger Anwendungen spielen Eigenschaften wie Fehlerfreiheit, Zuverlässigkeit, Integrität der Benutzerdaten, Systemantwortzeiten oder der effiziente Umgang mit Speicherplatz eine bedeutende Rolle. Diese Eigenschaften können unter dem Begriff *nicht-funktionale Laufzeiteigenschaften*<sup>1</sup> (NFE, engl. NFR) zusammengefasst werden. Sie gehören zur großen Gruppe der Softwarequalitäten und zeichnen sich dadurch aus, dass sie keine eigene Funktionalität bedingen, sondern Beschränkungen der durch die Software angebotenen Funktionalität definieren. Sind diese Eigenschaften zudem durch konkret messbare Werte ausdrückbar (z.B. „Das System muss in 98% der Anfragen fehlerfrei reagieren.“ oder „Die Reaktionszeit des Systems darf max. 3s betragen.“) handelt es sich um *quantitative* NFE. Nicht-funktionale Laufzeiteigenschaften geben ein Maß dafür an, wie gut funktionale Eigenschaften in einem laufenden System erfüllt werden. Sie machen daher den Wert eines Systems aus Sicht der Nutzer aus. Dadurch unterscheiden sie sich nach [MB01] von nicht-funktionalen Entwicklungszeiteigenschaften, wie Wiederverwendbarkeit und Wartbarkeit. Diese haben die Qualität der Softwareentwicklung zum Inhalt und damit Ziele der Organisation, welche das System entwickelt.

Bei der Entwicklung der hier betrachteten kritischen Softwaresysteme muss daher auch die Umsetzung und Erfüllung der nicht-funktionalen Laufzeitanforderungen systematisch und vor allem durchgängig durch den Softwareentwicklungsprozess unterstützt werden. Im Gegensatz zur Funktionalität eines Softwareprodukts, deren Umsetzung und Sicherstellung in gängigen Prozessmodellen sehr gut dokumentiert und damit in die Entwicklung integriert ist, ist die systematische Integration von Aktivitäten zur Umsetzung nicht-funktionaler Eigenschaften im gegenwärtigen Stand der Technik nicht gewährleistet und wird auch in der Forschung nur wenig thematisiert. Existierende Ansätze definieren Prozessmodelle entweder auf einer allgemeingültigen Abstraktionsebene, indem alle Qualitätseigenschaften abgedeckt werden oder es werden sehr detaillierte Prozesse spezifiziert, welche dann aber nur eine spezifische Qualitätseigenschaft beachten. Die Unterstützung von Leistungseigenschaften im Software Performance Engineering [SG02] sowie für Zugriffssicherheit in [Pop05] sind Beispiele für sehr spezifische Vorgehensmodelle,

---

<sup>1</sup>Synonym wird in dieser Arbeit der Begriff „laufzeitkritische nicht-funktionale Eigenschaft“ verwendet.

in denen konkrete Beschreibungs- und Verifikationstechniken vorgeschrieben sind. RUP [Kru03] beschreibt dagegen einen Subworkflow, in dem allgemeine Hinweise zum Testen gegeben werden und ist damit ein Beispiel für eine allgemeingültige Vorgehensspezifikation. Weiterhin existieren Ansätze, die nur eine Phase der Softwareentwicklung, wie z.B. die Anforderungsanalyse untersuchen (z.B. [CNYM00]). Außer den bereits angesprochenen Abstraktionsstufen, der allgemeinen und auf eine spezifische Eigenschaft angepassten Modelle, gibt es eine weitere, auf der Prozesse der Ausführungsebene beschrieben sind. Diese enthalten vorwiegend Beschreibungen organisatorischer Randbedingungen einer konkreten Projektdurchführung.

Der Funktionsumfang und damit die Komplexität der oben genannten Systeme wird weiter wachsen und dadurch die Systementwickler vor neue Herausforderungen stellen. Eine dieser Herausforderungen ist es, für kritische Anwendungen sorgfältig definierte Softwareentwicklungsprozesse bereitzustellen, um fehlerhaftes Verhalten der Anwendung frühzeitig und damit kostengünstig zu vermeiden [Som07]. Dass die Qualität von Software zu einem wesentlichen Teil durch die Qualität des Softwareentwicklungsprozesses bestimmt wird, ist eine der Motivationen warum Firmen Prozessbewertungstechniken, wie CMMI oder SPICE einsetzen. Diese bewerten Prozesse anhand verschiedener Kriterien und ordnen sie einer Ausbaustufe zwischen 0 und 5 zu. In der Praxis wird derzeit von den meisten Unternehmen die Stufe 2 bzw. 3 angestrebt [HDHM06]. In beiden Stufen wird sowohl die Definition und Dokumentation von Vorgehensmodellen als auch der Einsatz von Validierungs- und Verifikationstechniken gefordert.

### 1.1 Problemstellung

Zur Integration von Techniken zur Umsetzung nicht-funktionaler Laufzeiteigenschaften in Prozessmodellen (wie sie in CMMI und SPICE gefordert ist) gibt es demnach eine breite Vielfalt und Unübersichtlichkeit von Ansätzen. Warum gibt es aber bisher keine bekannten Ansätze zur Vereinheitlichung?

- Das Hauptproblem liegt in der *Vielschichtigkeit und Veränderlichkeit nicht-funktionaler Eigenschaften*. Es ist offensichtlich, dass die Forderung nach einer gewissen Antwortzeit und die nach Integrität der Daten unterschiedliche Auswirkungen auf die Umsetzung, damit auch auf die Qualitätssicherung und im Endeffekt auf den konkreten Entwicklungsprozess haben. Außerdem verändern sich nicht-funktionale Eigenschaften innerhalb des Entwicklungszyklus. Nicht-funktionale Anforderungen in der Anforderungsanalyse werden schrittweise zu funktionalen Operationen und Algorithmen zur Laufzeit. Zur Einhaltung von Antwortzeiten muss beispielsweise Rechenzeit und Speicherplatz allokiert werden.
- Eine weitere Problematik bilden die *verschiedenen Abstraktionsebenen* auf denen derartige Vorgehensmodelle definiert sind. Vorgehensmodelle werden weit vor dem eigentlichen Projektstart definiert und dann an die Gegebenheiten des Projektes angepasst. Allerdings sind, wie bereits diskutiert, konkrete Aktivitäten und zugehörige Werkzeuge zur Sicherstellung der Einhaltung von nicht-funktionalen

Eigenschaften abhängig von deren konkreter Ausprägung. Teilweise können Qualitätsanforderungen erst spät in der Anforderungsanalyse und damit nach Projektstart konkretisiert werden. Die Art der Anforderung hat aber Auswirkungen auf die Prozessdefinition. Diese Problematik wird in kleinen Projekten durch die Kommunikation der Prozessbeteiligten entschärft. In größeren Projekten jedoch sollten nicht-funktionale Anforderungen während der gesamten Projektlaufzeit durchgängig durch Prozessintegration unterstützt werden [Bor04], was durch die genannten Eigenschaften nicht von vornherein in allen Schritten der Vorgehensmodellerstellung möglich ist. Es wird vielmehr ein dynamisch-adaptives Vorgehen zur Bereitstellung des Entwicklungsprozesses benötigt.

- Daraus ergibt sich das Problem der *unzureichenden Unterstützung der Wiederverwendung* von Prozessbeschreibungen. Da es aus den genannten Gründen kein allgemeingültiges Vorgehensmodell geben kann, welches konkrete Umsetzungstechniken zur Entwicklung laufzeitkritischer Software beschreibt, existieren derzeit auch nur eingeschränkte Möglichkeiten der Wiederverwendung in weiteren Projekten. Die Verbindung und damit die Wiederverwendung von Prozessmodellen über die Abstraktionsebenen hinweg wird von Osterweil bereits in [Ost05] gefordert, ist aber im Bereich der Vorgehensmodelle für kritische Software aus den genannten Gründen noch nicht erfolgt.
- Durch die Integration von unterstützenden Methoden und Techniken zur Umsetzung der nicht-funktionalen Laufzeiteigenschaften wird der Prozess der Softwareentwicklung hinsichtlich *Kommunikationsstruktur und -inhalten komplexer*. Im Bereich der Werkzeuge für die Unterstützung der modellgetriebenen Entwicklung gibt es bereits erste Bestrebungen, den Entwickler durch die Automatisierung von Prozessschritten zu unterstützen. Der Einsatz dieser automatisierten Werkzeugketten ist allerdings bei den Prozessen, welche für die Realisierung nicht-funktionaler Laufzeiteigenschaften eingesetzt werden, unzureichend. Im Gegensatz zur modellgetriebenen Entwicklung muss im Kontext nicht-funktionaler Eigenschaften eine heterogene Landschaft aus Techniken, Werkzeugen und Rollen gesteuert werden. Dies ist einerseits bedingt durch die Integration von Validierung und Verifikation der nicht-funktionalen Anforderungen, andererseits sind zusätzlich Modelle bzw. Spezifikationen zur Beschreibung nicht-funktionaler Eigenschaften zu managen.

## 1.2 Zielstellung

Das generelle Ziel der Arbeit ist die Bereitstellung eines erweiterten Prozess- bzw. Vorgehensmodells sowie eines Prozess-Frameworks zur Unterstützung der systematischen Behandlung von nicht-funktionalen Laufzeiteigenschaften bei der Entwicklung laufzeitkritischer Systeme. Dieses Ziel umfasst sowohl eine *Methodik* als auch die entsprechende *Werkzeugunterstützung* für die Bereitstellung und Nutzung dieser Prozesse über die Abstraktionsebenen hinweg. Folgende Aufgaben sind dafür zu lösen:

## 1 Einleitung

1. *Spezifikation notwendiger Erweiterungen von Vorgehensmodellen zur systematischen Unterstützung laufzeitkritischer Eigenschaften:* Als Ausgangsbasis für die Vorgehensmodellierung ist die Untersuchung der existierenden Methodendlandschaft im Bereich der Umsetzung nicht-funktionaler Laufzeiteigenschaften notwendig. Daraus sind die notwendigen Konzepte bezüglich der Erweiterungen von allgemeinen Vorgehensmodellen für die systematische Umsetzung derartiger Eigenschaften zu extrahieren.
2. *Die Entwicklung einer Methodik zur Modellierung von Vorgehensmodellen für die Entwicklung laufzeitkritischer Systeme unter Beachtung der systematischen Behandlung nicht-funktionaler Laufzeitanforderungen* mit folgenden Eigenschaften:
  - Unterstützung der Wiederverwendung von Vorgehensmodellen und deren Fragmenten
  - Verbindung der Abstraktionsstufen von Vorgehensmodellen durch Verfeinerung entsprechend der Konkretisierung der nicht-funktionalen Eigenschaften während der Systementwicklung und damit Unterstützung einer adaptiven Erstellung des Entwicklungsprozesses
  - Unterstützung der Modellierung von Werkzeugketten
3. *Die Entwicklung der technischen Grundlagen und einer Werkzeugunterstützung für die Bereitstellung und Ausführung der erweiterten Vorgehensmodelle zur Entwicklung kritischer Systeme.* Dazu gehören folgende Teilaufgaben:
  - Schaffung struktureller Klarheit durch eine einheitliche Begriffswelt
  - Unterstützung der Entwickler durch Bereitstellung von Methodenwissen zur Umsetzung nicht-funktionaler Eigenschaften (Anleitungen, Einordnung in Gesamtverfahren, Zuordnung zu Artefakten) innerhalb einer Bibliothek
  - Unterstützung von Managern und Prozessingenieuren bei der Vorgehensmodellierung durch Werkzeugunterstützung für die bereitzustellende Methodik
  - Prozessunterstützung für kollaboratives Arbeiten durch eine entsprechende Ausführungsumgebung bei der Umsetzung der Werkzeugketten, da in die Entwicklung kritischer Systeme verschiedene Experten involviert sind

Der Fokus der Arbeit liegt auf quantitativen (also meßbaren) laufzeitbasierten Eigenschaften von Systemen innerhalb einer modellgetriebenen Entwicklung. Die Ergebnisse der Arbeit werden im Kontext anderer nicht-funktionaler Systemeigenschaften bzw. anderer nicht-modellbasierter Vorgehen diskutiert.

Zur Überprüfung der Arbeitsergebnisse auf die formulierten Ziele wurde ein Kriterienkatalog für die Validierung der Ergebnisse aufgestellt. Dieser wird in den entsprechenden Kapiteln zur Bewertung und Einordnung der Ergebnisse der Arbeit genutzt. Er beinhaltet folgende Kriterien:

1. Methodische Grundlagen:  
Die Validierung der zu entwickelten Methode zur Modellierung und Konkretisierung von Vorgehensmodellen für laufzeitkritische Systeme erfolgt anhand einer

durchgängigen Fallstudie. Damit soll die vollständige Abdeckung des Modellierungsprozesses eines derartigen Vorgehens demonstriert werden. Außerdem soll gezeigt werden, dass mittels der Methode zwei bekannte Vorgehensmodelle in der Domäne der kritischen Softwareentwicklung erstellt werden können. Die entwickelte Methode muss dazu folgende Eigenschaften besitzen:

- Einheitliche Modellierungssprache für Prozessmodelle auf Basis bekannter Standards. Hier ist die Frage zu beantworten, inwieweit deren Modellierungstechniken ausreichend sind.
- Bereitstellung von spezifischen Prozessbausteinen zur Umsetzung laufzeitbasierter nicht-funktionaler Eigenschaften
- Durchgängigkeit der Vorgehensmodellierung von abstrakten Prozessbausteinen bis zur Ausführungsunterstützung
- Mögliche Integration mit anderen Softwareentwicklungsprozessen

### 2. Technische Grundlagen und Werkzeugunterstützung

Die Validierung der technischen Grundlagen und der Werkzeugunterstützung erfolgt auf Basis einer prototypischen Implementierung eines Prozess-Frameworks. Dieses muss mindestens soweit entwickelt sein, dass die Durchgängigkeit der Methodik vom abstrakten Vorgehensmodell auf der Makroprozessebene bis hin zur Ausführung innerhalb eines konkreten Projekts demonstriert werden kann. Dafür müssen folgende Kriterien erfüllt sein:

- Bereitstellung der spezifischen Prozessbausteine innerhalb einer prototypischen Implementierung als Prozessbibliothek
- Durchgängige Unterstützung der entwickelten Methode als Machbarkeitsbeweis
- Verbindung von Prozesserstellung und Prozessausführung um zu gewährleisten, dass das Prozesswissen für die Entwickler präsent ist

## 1.3 Ergebnisse

Die Arbeit stellt die *Methodik der kontextbasierten Anpassung von Vorgehensmodellen für die Entwicklung laufzeitkritischer Softwaresysteme* vor. Die Methodik basiert auf folgenden Konzepten: Methodenlandschaft in Form von NFE-Prozessmustern für die Erweiterung von Vorgehensmodellen zur Entwicklung kritischer Software, Wiederverwendung und Komposition, Verfeinerung von Vorgehensmodellen zur Verbindung von Makro- und Mikroprozessen sowie der Prozessausführung.

Damit stellt die Arbeit erstmals einen systematischen Ansatz zur konsequenten Umsetzung laufzeitkritischer nicht-funktionaler Anforderungen durch Entwicklungsprozesse vor. Die wesentlichen Herausstellungsmerkmale der Arbeit sind:

1. *Systematisierung der Unterstützung von laufzeitbasierten nicht-funktionalen Eigenschaften durch Entwicklungsprozesse:* Auf der Basis einer umfassenden Analyse des

## 1 Einleitung

Stands der Technik auf dem Gebiet der Prozessunterstützung nicht-funktionaler Laufzeiteigenschaften, wurden 13 NFE-Prozessmuster identifiziert und bereitgestellt. Diese verallgemeinern Erweiterungen von Softwareprozessen, die für die Umsetzung nicht-funktionaler Eigenschaften notwendig sind.

2. *Entwicklung einer Methode und der zugehörigen Werkzeugunterstützung zur Arbeit mit dem durch NFE-Prozessmuster erweiterten Vorgehensmodell:* Um sowohl die Wiederverwendbarkeit von Vorgehensmodellen zu ermöglichen, als auch die Erfüllung der nicht-funktionalen Eigenschaften zu gewährleisten, wurde die PROKRIS<sup>2</sup>-Methodik als neues Entwicklungsmodell für die dynamisch-adaptive Bereitstellung der notwendigen Vorgehensmodelle entwickelt. Unterschieden werden die drei Anpassungsstufen abstraktes, NFE-spezifisches und ausführbares Vorgehensmodell. Die Kernidee ist die Wiederverwendung und konsequente Verfeinerung von Vorgehensmodellen über diese Abstraktionsebenen hinweg. Dadurch wird die flexible Erweiterung und Anpassung von Vorgehensmodellen für die systematische Umsetzung nicht-funktionaler Laufzeiteigenschaften möglich.
3. *Durchgängigkeit der Prozessmodellierung bis zur Ausführungsunterstützung:* Die Arbeit stellt außerdem die Verbindung zur Ausführungsebene der Prozesse her. Dies ermöglicht die Bereitstellung von prozessunterstützten Werkzeugumgebungen, welche individuell an die Projektbedingungen angepasst sind. Mit der in dieser Arbeit vorgestellten Lösung können die konkreten Ressourcen einer Organisation (Personen, Werkzeuge und Daten) erst zum spätest möglichen Zeitpunkt fest zugeordnet und die Prozessbeschreibung so lange wie möglich diesbezüglich flexibel eingesetzt werden.

Die unterstützende Werkzeugumgebung ist das *PROKRIS-Framework*, welches aus folgenden Basiskomponenten aufgebaut ist:

- *PROKRIS-Prozessbibliothek:* In dieser werden sowohl die NFE-Prozessmuster als auch die Prozessfragmente der NFE-spezifischen Vorgehensmodelle bereitgestellt. Dazu gehören Aufgaben- und Rollenbeschreibungen und ihre Verknüpfungen zu Artefakten. Die Erstellung und Pflege der Prozessbibliothek unterliegt einem eigenen Prozess, auf den kurz eingegangen wird.
- *PROKRIS-Prozesskomposer:* Der Prozesskomposer unterstützt den Algorithmus der Konkretisierung von Vorgehensmodellen und damit deren Anpassung an die systematische Unterstützung für verschiedene nicht-funktionale Eigenschaften bzw. an spezifische Projektbedingungen.
- *Ausführungskomponente:* Das erweiterte Vorgehensmodell kann auf der Beschreibungsebene als Basis für ein Handbuch genutzt werden. Außerdem kann es an spezifische Organisationsstrukturen angepasst und mittels einer Ausführungsumgebung ausgeführt werden. Den Entwicklern stehen so individuell angepasste integrierte

---

<sup>2</sup>PROzessunterstützung für die Entwicklung laufzeit-KRItischer Softwaresysteme

Werkzeugumgebungen zur Verfügung, die mit der Prozessbeschreibung gesteuert werden.

In diesem Zusammenhang wurden weitere wichtige Ergebnisse erzielt. So wurde das Software Process Engineering Metamodel der OMG (SPEM 2.0) um wesentliche Aspekte erweitert. Dazu zählen: bessere Unterstützung der Komposition durch einen Filteralgorithmus basierend auf einer Facettenklassifikation zur Beschreibung des Projektkontexts, Verbindung verschiedener Abstraktionsebenen von Prozessen, Modellierung von Prozessressourcen und Abbildungsmöglichkeiten auf eine Ausführungssprache.

Des Weiteren war es für die Strukturierung der PROKRIS-Bibliothek notwendig, eine Klassifikation der nicht-funktionalen Eigenschaften zu erstellen. Auf der Grundlage der Analyse existierender Klassifikationen wurde eine Facettenklassifikation nicht-funktionaler Eigenschaften erstellt, welche die Differenzierung von Typ, Repräsentation und Erfüllung ermöglicht, sowie eine entsprechende Taxonomie für die nicht-funktionalen Eigenschaften vorgestellt.

Auf Seiten der Werkzeugunterstützung wurde die Standardarchitektur für Workflowumgebungen der WfMC [Wor95] zur Unterstützung von Softwareentwicklungsprozessen erweitert. Die definierten Erweiterungskonzepte ermöglichen die Instanziierung einer auf das jeweilige Entwicklungsprojekt abgestimmten prozessbasierten integrierten Entwicklungsumgebung.

Außerdem sind die Erweiterungen der Architektur allgemeiner Prozess-Frameworks durch das PROKRIS-Framework auch für andere Anwendungsfelder einsetzbar. Sowohl der Filteralgorithmus des PROKRIS-Komposers als auch die Übersetzung in die ausführbare Spezifikationssprache sowie die Ausführungsumgebung sind so entworfen, dass diese unabhängig vom Anwendungskontext der nicht-funktionalen Eigenschaften anwendbar sind.

Darüber hinaus leistet die Entwicklung des PROKRIS-Ansatzes einen wesentlichen Beitrag zur systematischen Modellierung und Ausführung von Softwareentwicklungsprozessen nach der von Osterweil bereits 1987 aufgestellten These „Software Processes are Software too“ [Ost87]. Ferner wird durch die Bereitstellung der Ausführungsumgebung auf der Grundlage der NFE-spezifischen Vorgehensmodelle eine Dokumentierbarkeit der Entwicklung möglich. Dies hat direkte Auswirkungen auf die erreichbare CMMI-Stufe. Außerdem ist die Entwicklung anhand eines festgelegten Prozesses sowie dessen Dokumentierbarkeit unabdingbare Voraussetzung für die Zertifizierung von Software.

## 1.4 Aufbau der Arbeit

Nach einer Einführung in die für das Verständnis der Arbeit notwendigen Grundlagen in Kapitel 2, werden in Kapitel 3 die Herausforderungen, welche nicht-funktionale Eigenschaften an Vorgehensmodelle und Prozessunterstützung stellen, anhand eines Beispielpjekts diskutiert. Außerdem stellt dieses Kapitel die Idee des PROKRIS-Frameworks vor und gibt einen Überblick über die Architektur und die Einsatzszenarien des PROKRIS-Frameworks. Die Konzepte der Bereitstellung eines um NFE-Prozessmuster erweiterten Vorgehensmodells zur Entwicklung laufzeitkritischer Software

## 1 Einleitung

werden in Kapitel 4 vorgestellt. Kapitel 5 behandelt die Grundlagen der neuen Methodik der Prozessmodellierung. Die folgenden drei Kapitel stellen im Detail die einzelnen Bausteine des Frameworks vor: Kapitel 7 den Aufbau der PROKRIS-Prozessbibliothek sowie die zugrundeliegenden Metamodellkonzepte, Kapitel 8 die Vorgehensmodellkomposition und -verfeinerung und Kapitel 9 die Ausführungsunterstützung. Die Evaluierung der Ergebnisse wird in Kapitel 10 diskutiert. Dazu zählen die prototypische Umsetzung des Ansatzes innerhalb einer Werkzeugumgebung, die Anwendung der Methode in zwei Fallstudien und der Entwicklung einer laufzeitkritischen Beispielanwendung sowie die Diskussion des Einsatzes in Verbindung mit bekannten Vorgehen wie dem V-Modell XT und Rational Unified Process (RUP). Verwandte Forschungsarbeiten werden in Kapitel 4 (NFE-Unterstützung in bestehenden Vorgehensmodellen) und in Kapitel 11 diskutiert. Die Arbeit schließt mit einer Zusammenfassung der Ergebnisse sowie einem Ausblick auf mögliche weiterführende Arbeiten in Kapitel 12.

In der Arbeit wird der im SuReal-Projekt<sup>3</sup> [SuR] konzipierte Entwicklungsprozess, welcher auf die systematische Umsetzung von Realzeiteigenschaften sicherheitskritischer Systeme spezialisiert ist, als durchgängiges Beispiel verwendet. Dies bedingt die Fokussierung der Erklärungen und Diskussionen auf formale Verifikationstechniken zur Überprüfung der Erfüllung der nicht-funktionalen Anforderungen. Dieser Ansatz ist auf andere Prüftechniken, wie Testen und Simulieren übertragbar, was an den jeweiligen Stellen der Arbeit entsprechend diskutiert wird.

---

<sup>3</sup>Forschungsprojekt „SuReal-Sicherheitsgarantien unter Realzeitanforderungen“, gefördert durch das BMBF, Laufzeit: Mai 2006 bis April 2009



## 2 Grundlagen

Dieses Kapitel stellt die konzeptuellen Grundlagen der relevanten Fachgebiete vor. Es werden die in den folgenden Kapiteln benutzten Begriffe erläutert und verwendete Beschreibungsmittel vorgestellt. Relevante Fachgebiete sind nicht-funktionale Anforderungen an Softwaresysteme, Softwareentwicklungsprozesse, sowie die Softwarequalitätssicherung. Dabei ist vor allem das Gebiet der Softwareentwicklungsprozesse sehr breit gefächert. Das Kapitel beschränkt sich deshalb auf die relevanten Bereiche, wie die Modellierung von Softwareprozessen, Wiederverwendungsaspekte und die Ausführung von Prozessspezifikationen.

Da es sich selbst bei dieser Einschränkung um ein umfangreiches Spektrum an Grundlagen handelt, werden nur die wesentlichen Konzepte erläutert. Zu den einzelnen Abschnitten ist jeweils Literatur angegeben, die bei Interesse einen tieferen Einblick in das jeweilige Gebiet liefert. Einen Überblick über ähnliche Forschungsarbeiten gibt Kapitel 10.

### 2.1 Der Begriff „nicht-funktionale Eigenschaft (NFE)“

Um sich mit der Behandlung von nicht-funktionalen Anforderungen bzw. Eigenschaften in Softwareentwicklungsprozessen zu beschäftigen, ist es unerlässlich zu definieren, was unter einer nicht-funktionalen Anforderung zu verstehen ist. Dazu wird zuerst geklärt, was eine Anforderung ist. [Poh07] definiert eine Anforderung (requirement) nach [IEE90] folgendermaßen:

Eine **Anforderung** ist:

- (1) Eine Bedingung oder Eigenschaft, die ein System oder eine Person benötigt, um ein Problem zu lösen oder ein Ziel zu erreichen.
- (2) Eine Bedingung oder Eigenschaft, die ein System oder eine Systemkomponente aufweisen muss, um einen Vertrag zu erfüllen oder einem Standard, einer Spezifikation oder einem anderen formell auferlegten Dokument zu genügen.
- (3) Eine dokumentierte Repräsentation wie in (1) oder (2) definiert.

Diese Definition verdeutlicht, dass es keine substantielle Unterscheidung in *Anforderungen* an ein System und *Eigenschaften* eines Systems gibt. Mit dem Begriff Anforderung grenzt man jene Eigenschaften eines Systems ab, welche von außen, z.B. durch einen Nutzer, vom System gefordert werden. Anforderungen sind demnach eine besondere Gruppe von Eigenschaften, deren Erfüllung durch das System zu prüfen bzw. zu

beweisen ist. Da es sich aber immer um die Spezifikation von Eigenschaften handelt, können sowohl Eigenschaften als auch deren spezielle Form der Anforderungen mit den gleichen Mitteln beschrieben werden [Zsc07].

Anforderungen bzw. Eigenschaften können in funktionale und nicht-funktionale Eigenschaften unterschieden werden. Funktionale Anforderungen werden überall ähnlich definiert, beispielsweise nach [Som07] wie folgt:

Eine **funktionale Anforderung** definiert Aussagen über eine vom System bzw. von einer Systemkomponente bereitzustellende Funktion, einen bereitzustellenden Service oder eine Reaktion des Systems auf bestimmte Eingaben.

Die Definition des Begriffs „nicht-funktionale Eigenschaft“ erfolgt in der Literatur unter verschiedenen Begriffen. So wird auch von extra-funktionalen Eigenschaften, Software-Qualität oder Qualitätsattributen gesprochen. Die Grundaussage dieser Definitionen ist jedoch immer ähnlich. Eine der Definitionen liefert Sommerville [Som07]. Er definiert eine nicht-funktionale Eigenschaft folgendermaßen:

**Nicht-funktionale Anforderungen** sind Beschränkungen der durch das System angebotenen Dienste und Funktionen. Das schließt Zeitbeschränkungen, Beschränkungen des Entwicklungsprozesses und einzuhaltende Standards ein. Nicht-funktionale Anforderungen beziehen sich oft auf das ganze System und gewöhnlich nicht auf einzelne Systemfunktionen oder Dienste.

Diese Definition ist insofern sehr gut, als sie drei grundlegende Aussagen über das Wesen nicht-funktionaler Eigenschaften in einer Definition zusammenfasst. Die *erste Aussage* ist, dass es sich bei nicht-funktionalen Eigenschaften prinzipiell immer um Beschränkungen handelt. Dies ist auch die wesentliche Abgrenzung von einer funktionalen Eigenschaft. Funktionale Eigenschaften zeichnen sich immer durch einen booleschen Charakter aus. Sie sind entweder erfüllt oder nicht. Für nicht-funktionale Eigenschaften ist die Erfüllung dagegen optimierbar und es kann der Grad der Erfüllung angegeben werden [Zsc07].

Eine *zweite Aussage* dieser Definition ist, dass verschiedene Klassen nicht-funktionaler Eigenschaften existieren. Sommerville unterscheidet in seiner, als Erweiterung zu obiger Definition, aufgestellten Klassifikation in Produktanforderungen (Product requirements), Unternehmensanforderungen (Organisational requirements) und externe Anforderungen (External requirements) (Abb. 2.1). Ähnliche Unterscheidungen werden auch von anderen Autoren getroffen. [Poh07] definiert beispielsweise von vornherein zwei Begriffe: Qualitätsanforderung und Rahmenbedingung.

Eine **Qualitätsanforderung** definiert eine qualitative Eigenschaft des Systems oder einzelner Funktionen des Systems.

Eine **Rahmenbedingung** ist eine organisatorische oder technologische Anforderung, die die Art und Weise einschränkt, wie ein Produkt entwickelt wird.

## 2.1 Der Begriff „nicht-funktionale Eigenschaft (NFE)“

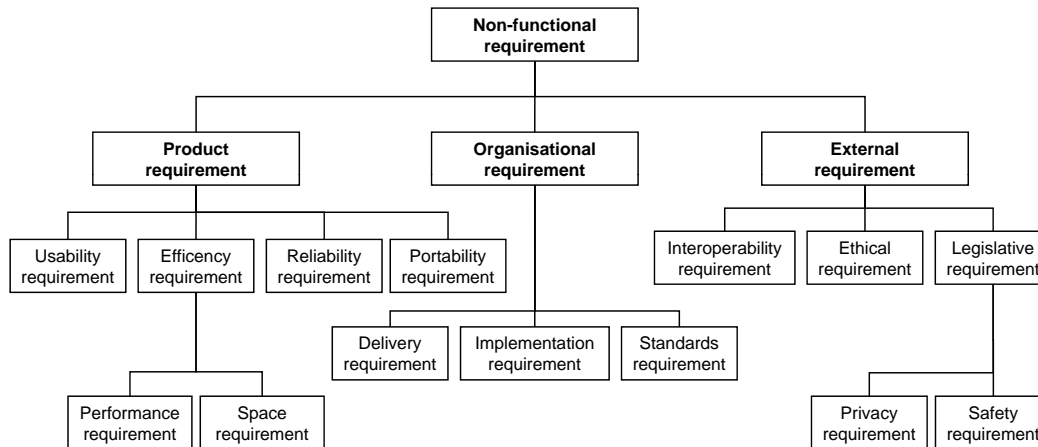


Abbildung 2.1: Klassifikation nicht-funktionaler Eigenschaften nach [Som07]

Er unterteilt also begrifflich in solche Anforderungen, die das Produkt an sich und solche, die den Umgang mit dem Produkt während der Entwicklung, Wartung oder Einsatz betreffen. Malan und Bredemeyer führen in [MB01] eine ähnliche Unterteilung in das lauffähige System und die Arbeitsprodukte, die bei der Entwicklung des Systems anfallen, ein. Diesen werden die *Run-time Qualities (Laufzeiteigenschaften)* bzw. die *Development-time Qualities (Entwicklungszeiteigenschaften)* zugeordnet. Laufzeiteigenschaften machen den Wert eines Systems aus Sicht der Nutzer aus. Um eine Bewertung der Eigenschaften zu ermöglichen, muss das Systemverhalten mit seinen einzelnen Funktionen beobachtbar und messbar gemacht werden. Im Gegensatz dazu verkörpern Entwicklungszeiteigenschaften die Ziele der Organisation, welche das System entwickelt. Sie haben die Qualität der Softwareentwicklung zum Inhalt. Sie repräsentieren daher in erster Linie den Entwicklungsaufwand, der für aktuelle und zukünftige Projekte zu betreiben ist bzw. zu betreiben sein wird.

Trotz der verschiedenen Begrifflichkeiten erfolgt die Unterteilung nicht-funktionaler Eigenschaften demnach in produktbezogene und prozessbezogene Eigenschaften. Am Besten beschreiben dies die Begriffe von Malen und Bredemeyer, weshalb diese Arbeit ebenfalls in Laufzeit- und Entwicklungszeiteigenschaften unterteilt. Tabelle 2.1 ordnet beispielhaft bekannte nicht-funktionale Eigenschaften in die beiden Klassen ein.

Die *dritte Aussage* in Sommervilles Definition bezieht sich auf den Gültigkeitsbereich nicht-funktionaler Anforderungen. Seine Aussage ist, dass sich diese Anforderungen meist auf das Gesamtsystem beziehen. Für die Entwicklung und Überprüfung von derartigen Eigenschaften ist es unerlässlich, nicht-funktionale Anforderungen genauso zu dekomponieren und in Eigenschaften von einzelnen Diensten, Funktionen oder Komponenten herunterzubrechen, wie es auch für funktionale Eigenschaften getan wird. Dabei handelt es sich nicht immer um eine 1:1-Abbildung. So besteht eine allgemeine nicht-funktionale Anforderungen „schnell“ an eine Videoübertragung aus mehreren Eigenschaften (z.B. Größe der Framerate und maximale Verzögerung) des dazu bereitzustellenden funktio-

Klasse	Eigenschaften
Run-time Qualities (Laufzeiteigenschaften)	Usability
	Configurability
	Supportability
	Correctness
	Reliability
	Availability
	Quality of Service
	Safety
	Operational scalability
Development-time Qualities (Entwicklungszeiteigenschaften)	Localizability
	Modifiability/Extensibility
	Evolvability
	Composability
	Reusability

Tabelle 2.1: Klassifikation nicht-funktionaler Eigenschaften nach [MB01]

nalen Dienstes [RZ04a].

Pohl [Poh07] definiert noch einen weiteren Begriff, die unterspezifizierte Anforderung.

Eine **unterspezifizierte Anforderung** ist vage definiert und lässt dadurch mehrere Interpretationen zu. Da unterspezifizierte Anforderungen nicht objektiv überprüft werden können, müssen sie durch funktionale Anforderungen und ggf. Qualitätsanforderungen konkretisiert werden.

Diese Aussage verdeutlicht ein grundlegendes Problem bei der Behandlung nicht-funktionaler Produkteigenschaften, die Schwierigkeit der eindeutigen Zuordnung von Eigenschaften zur funktionalen bzw. nicht-funktionalen Gruppe. So ist z.B. eine Benutzeranforderung, die die Sicherheit eines Systems betrifft, zunächst einmal eine nicht-funktionale Eigenschaft. Analysiert man diese Anforderung aber genauer, kann sie zu einer eindeutig funktionalen Umsetzung führen, indem ein Mechanismus zur Benutzerauthentifizierung im System gefordert wird. Ein anderes Beispiel sind Zeiteigenschaften. Diese können nur garantiert werden, wenn zur Laufzeit auch genügend CPU-Zeit und Speicherplatz bereitgestellt wird, was wiederum rein funktionale Eigenschaften sind [APRZ03]. Es kommt also immer auf die Abstraktionsebene an, ob eine Eigenschaft als nicht-funktional oder funktional anzusehen ist. Im Endeffekt spiegeln sich nahezu alle Produkt- oder besser Laufzeiteigenschaften eines Systems auf einer umsetzungsnahen Abstraktionsebene als Funktionalität wieder.

Betrachtet man nur die Klasse der Laufzeiteigenschaften in Tabelle 2.1, stellt man fest, dass sich darin zwei weitere Klassen unterscheiden lassen, *quantitative* und *qualitative* Eigenschaften. Quantitative Eigenschaften können durch die Definition und Überprüfung einer Maßangabe (measurement) verifiziert werden [Gli07]. Zu diesen Eigenschaften zählen zum Beispiel „Quality of Service“ und „Safety“. Qualitative Eigenschaften können

dagegen nicht direkt verifiziert werden. Ihre Verifikation erfolgt entweder subjektiv durch Nutzer, welche das System oder Prototypen davon bewerten oder indirekt über Metriken. Dazu zählen beispielsweise „Usability“ und „Configurability“. Diese Arbeit beschränkt sich zunächst auf die Gruppe der quantitativen Eigenschaften und damit der direkten Behandlung der Laufzeiteigenschaften.

### Zusammenfassung

Zusammenfassend lässt sich als Grundlage für die vorliegende Arbeit folgendes schließen:

- Eigenschaften, die von einem System, einer Komponente oder einer Funktion gefordert werden, bezeichnet man als Anforderung. Es handelt sich also bei Anforderungen um eine spezielle Form von Eigenschaften.
- Nicht-funktionale Eigenschaften können in Laufzeit- und Entwicklungszeiteigenschaften unterteilt werden.
- Nicht-funktionale Laufzeiteigenschaften beschreiben Beschränkungen von funktionalen Eigenschaften. Sie sind systemweit oder lokal bezüglich einer speziellen Funktionalität bewertbar.
- Die Erfüllung nicht-funktionaler Anforderungen durch Eigenschaften des Systems ist im Gegensatz zu funktionalen Eigenschaften nicht boolesch, sondern optimierbar.
- Nicht-funktionale Anforderungen können auf verschiedenen Abstraktionsebenen vorkommen: als grobe Nutzeranforderung, als Anforderung an das Gesamtsystem oder als Anforderung an einzelne Dienste. Diese Eigenschaften müssen zwischen den Abstraktionsebenen dekomponiert bzw. komponiert werden. Dabei kann es sich um eine 1:n-Abbildung zwischen Anforderungen und Systemeigenschaften handeln. Außerdem können aus nicht-funktionalen Eigenschaften auf niedrigeren Abstraktionsebenen der Systemumsetzung funktionale Eigenschaften entstehen.
- Als *Eingrenzung* beschäftigt sich die vorliegende Arbeit nur mit *quantitativen nicht-funktionalen Laufzeiteigenschaften*.

## 2.2 Klassifikation nicht-funktionaler Eigenschaften

Neben den bereits vorgestellten Klassifikationen nicht-funktionaler Eigenschaften nach Sommerville bzw. Malan/Bredemeyer existieren weitere. Im Kontext einer systematischen Prozessunterstützung solcher Eigenschaften werden einige ausgewählte Ansätze vorgestellt, um daraus Rückschlüsse für eine prozessrelevante Klassifikation zu ziehen. Einen Überblick über weitere Einteilungen gibt [Zsc07].

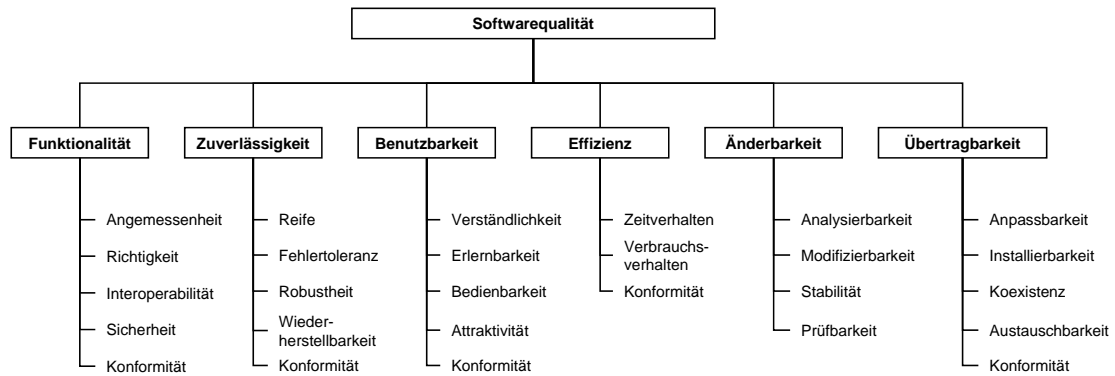


Abbildung 2.2: Qualitätsmodell nach ISO/IEC 9126

### Qualitätsmodell nach ISO/IEC 9126 (DIN 66272)

[ISO91]<sup>1</sup> stellt ein standardisiertes Qualitätsmodell zur Verfügung. Es behandelt ausschließlich Produktqualität (also keine Prozessqualität). Abbildung 2.2 zeigt die Klassifikation. Innerhalb des Standards sind die einzelnen Klassen eindeutig definiert. An dieser Stelle werden nur die Definitionen der Oberklassen aufgeführt.

*Funktionalität:* Implementierung und Ausführbarkeit der definierten funktionalen Anforderungen

*Zuverlässigkeit:* Fähigkeit der Software, ihr Leistungsniveau unter festgelegten Bedingungen über einen festgelegten Zeitraum zu bewahren

*Benutzbarkeit:* Aufwand, der zur Benutzung erforderlich ist, sowie individuelle Beurteilung der Benutzung durch eine festgelegte oder vorausgesetzte Benutzergruppe

*Effizienz:* Verhältnis zwischen dem Leistungsniveau der Software und dem Umfang der eingesetzten Betriebsmittel unter festgelegten Bedingungen

*Änderbarkeit:* Aufwand, der zur Durchführung vorgegebener Änderungen (Korrekturen, Verbesserungen und Anpassung an Änderungen von Anforderungen) notwendig ist

*Übertragbarkeit:* Eignung der Software, von einer Umgebung (organisatorische Umgebung, Hardware- oder Softwareumgebung) in eine andere übertragen zu werden

### COMQUAD - Klassifikation

Innerhalb des Forschungsprojekts COMQUAD [COM], [HZP<sup>+</sup>07] wurde basierend auf ([MB01], [Som07] und [RZ03]) eine sehr ausführliche Klassifikation nicht-funktionaler Eigenschaften aufgestellt. Ziel war dabei die Bereitstellung einer Bibliothek von Maßen nicht-funktionaler Eigenschaften, die an verschiedenen Stellen des Entwurfsprozesses

<sup>1</sup>Das beschriebene Qualitätsmodell ist identisch mit dem Modell der bekannten DIN 66272: Bewerten von Softwareprodukten - Qualitätsmerkmale und Leitfaden zu ihrer Verwendung. Die Norm wurde im Mai 2006 durch die ISO/IEC 9126 ersetzt.

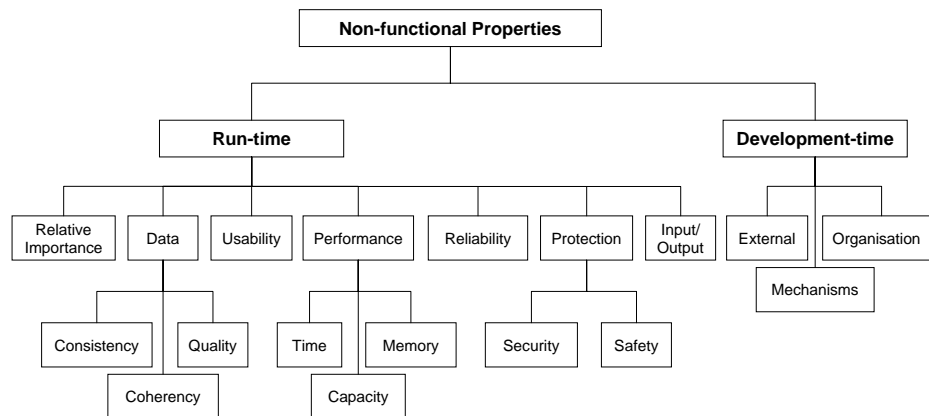


Abbildung 2.3: COMQUAD-Klassifikation nach [Ban04]

Verwendung finden kann. Der Einsatz der Maßbibliothek innerhalb eines Entwicklungsprozesses ist in [RZ07] dargestellt. Obwohl das Ziel ein anderes ist, wird die Klassifikation zum Aufbau der Struktur der Prozessbibliothek mit herangezogen, da sie sehr ausführlich ist. Andererseits ist der Aufbau der Prozessbibliothek ein logischer weiterer Ausbauschritt der in [RZ07] beschriebenen Prozessmethodik.

Die Klassifikation ist in Abbildung 2.3 dargestellt. Da sich diese Arbeit auf laufzeitbasierte NFE einschränkt, werden nur die Definitionen für Run-time aufgeführt. Die einzelnen Eigenschaftsklassen sind wie folgt definiert:

*Relative Importance (Relative Bedeutung):* Eigenschaften, die direkt durch Prioritäten beschrieben werden.

*Data (Daten):* Eigenschaften, die Merkmale von Datentypen oder Zusammenhänge zwischen Daten beschreiben.

*Usability (Nutzbarkeit):* Eigenschaften, die zum Ausdruck bringen, wie hoch der Aufwand für einen Nutzer ist, um mit einem System/Teilsystem zu arbeiten.

*Performance (Leistungsfähigkeit):* Eigenschaften, die das Verhalten eines Systems/Teilsystems bezüglich seiner Geschwindigkeit, Auslastung und seines Speicherbedarfes beschreiben.

*Reliability (Zuverlässigkeit):* Eigenschaften, die Ausfall- und Fehlverhalten eines Systems/Teilsystems verkörpern.

*Protection (Schutz):* Eigenschaften, die mit Schutzzielen in Verbindung stehen. Dabei wird die Schnittstelle zwischen System und Systemumfeld aus beiden Richtungen betrachtet.

*Input/Output (Eingabe/Ausgabe):* Eigenschaften zur Beschreibung von Aspekten eines Systems/Teilsystems in Verbindung mit Eingabe und Ausgabe.

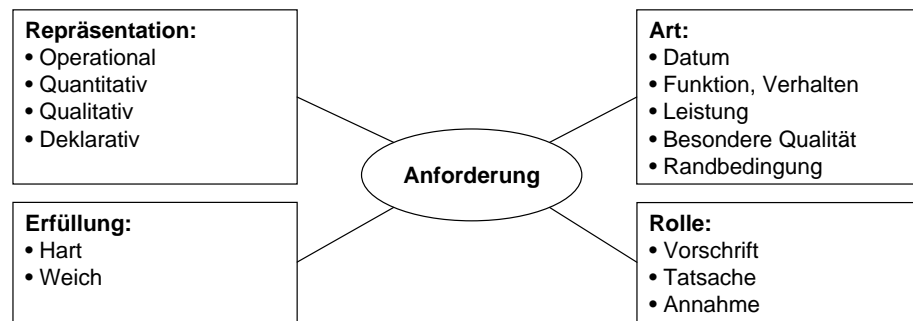


Abbildung 2.4: Facettenklassifikation nach Glinz [Gli05]

Für die weitergehende Beschäftigung mit dieser ausführlichen Klassifikation sei auf [Ban04] verwiesen, in der sowohl die vollständige Klassifikation beschrieben ist, als auch eine umfassende Taxonomie verschiedenartiger nicht-funktionaler Eigenschaften mittels dieser Klassifikation erstellt wurde.

### Facettenklassifikation nach Glinz

Traditionell werden, wie auch die bisher vorgestellten Ansätze zeigen, Systemanforderungen in funktionale und nicht-funktionale Eigenschaften unterteilt. Im Gegensatz dazu vertritt Glinz in seinen Arbeiten ([Gli05], [Gli07]) eine andere Auffassung einer Einteilung von NFE, auf die kurz eingegangen werden soll. Er unterteilt die Probleme der Behandlung nicht-funktionaler Anforderungen in Definitions-, Klassifikations- und Repräsentationsprobleme. Das Definitionsproblem wurde bereits diskutiert. Es sagt aus, dass keine eindeutige Definition des Begriffs der nicht-funktionalen Eigenschaft existiert. Das Klassifikationsproblem wurde ebenfalls bereits deutlich. Jeder der bisher vorgestellten Ansätze ist verschieden. Das Repräsentationsproblem beschreibt die Veränderung nicht-funktionaler Anforderungen während der Entwicklungszeit durch Operationalisierung zu Funktionen des Systems und damit zu funktionalen Eigenschaften. Des Weiteren existiert keine einheitliche Notation für nicht-funktionale Eigenschaften.

Zur Lösung des Klassifikations- sowie des Repräsentationsproblems schlägt Glinz eine vollständig andere Klassifikation von Systemanforderungen vor. Er verwendet zur Einordnung vier Kriterien, die allen Anforderungen und damit Systemeigenschaften gemein sind. Die Kriterien bilden eine Facettenklassifikation mit folgenden Facetten: *Art* (Kind), *Repräsentation* (Representation), *Erfüllung* (Satisfaction) und *Rolle* (Role). Abbildung 2.4 gibt einen Überblick über die Facetten und ihre Unterkategorien.

In [Gli07] wird dieser Ansatz verwendet, um verschiedene Eigenschaften innerhalb der Facette *Art* zu klassifizieren. Die entstehende Klassifikation unterscheidet sich nicht wesentlich von den bereits vorgestellten.

### Bewertung der Ansätze

Die Klassifikation nach Sommerville zeichnet sich durch große Detailliertheit aus. Allerdings werden die Klassen nur auf der ersten Hierarchieebene der Klassifikation definiert.



Die Nutzung der Klassifikation ist somit immer abhängig von einer eigenständig zu definierenden Bedeutung der einzelnen Klassen, die in den tieferen Hierarchieebenen liegen. Durch das Fehlen der konkreten Definitionen, stellt Sommervilles Klassifikation zwar einen guten Überblick zur Verfügung, eine eindeutige Benutzung ist aber nicht möglich.

Das Qualitätsmodell der ISO 9126 umfasst alle wesentlichen Aspekte der Produktqualität. Dazu zählen sowohl Laufzeit- als auch Entwicklungszeiteigenschaften. Der PROKRIS-Ansatz schränkt sich von vornherein auf die Behandlung von Laufzeiteigenschaften ein. Diese strikte Unterscheidung ist im Modell der ISO 9126 nicht vorhanden. So zählen zur Zuverlässigkeit sowohl Laufzeiteigenschaften, wie Fehlertoleranz, als auch Entwicklungszeiteigenschaften, wie Reife. Daher ist das Qualitätsmodell für den PROKRIS-Ansatz weniger interessant.

Als Ausgangspunkt für die COMQUAD-Klassifikation diente die Klassifikation nach Sommerville. Wesentliche Klassen wurden direkt übernommen. Die Klassifikation strukturiert die Laufzeiteigenschaften sehr detailliert. Dabei ist die Klassifikation unterhalb der ersten Hierarchieebene keinesfalls endgültig. Sowohl im Bereich der Laufzeit- als auch der Entwicklungszeiteigenschaften ist die Änderung der Klassifikation durch Einführung neuer Klassen möglich.

Die Facettenklassifikation nach Glinz weist zwei wesentliche Vorteile auf, welche auch für den PROKRIS-Ansatz relevant sind. Erstens bleibt, durch die strikte Trennung der Repräsentation von der Art der Eigenschaften, die Einordnung einer spezifischen Eigenschaft über die Verfeinerung innerhalb des Entwicklungszyklus hinaus die gleiche. Das bedeutet, wird aus einer nicht-funktionalen Eigenschaft auf Modellebene im Laufe der Entwicklung eine Funktionalität, ändert sich nur deren Repräsentationsfacette. Der zweite Vorteil ist, dass aus der Einordnung in die Facette Repräsentation direkt auf die Art der Verifikation geschlossen werden kann. So werden quantitative Eigenschaften durch Überprüfung der Maße der Eigenschaft verifiziert, wogegen deklarative Eigenschaften durch Reviewtechniken kontrollierbar sind.

Zusammenfassend lässt sich sagen, dass es kein Ansatz vermag, eine vollständige Klassifikation zu entwerfen, die alle Bereiche nicht-funktionaler Eigenschaften mit einer detaillierten Strukturierung abdeckt. Die Ursachen sind vor allem in der Vielfalt und der Komplexität dieser Eigenschaften zu sehen.

## 2.3 Grundbegriffe von Entwicklungsprozessen

Um eine Einordnung existierender Prozesse, Aktivitäten, angewandeter Methoden und Techniken in Bezug auf die Unterstützung nicht-funktionaler Eigenschaften vornehmen zu können, ist es wichtig, die verschiedenen Abstraktionsebenen, auf denen die Definition von Vorgehens- und Prozessmodellen der Softwareentwicklung möglich ist, zu diskutieren. Außerdem gibt es bei der Betrachtung von Softwareentwicklungsprozessen eine große Begriffsverwirrung, was die Kommunikation darüber erschwert [ZGK04]. Daher ist es notwendig, die in der Arbeit verwendeten Grundbegriffe in diesem Kontext zu definieren.

### 2.3.1 Vorgehens- und Prozessmodelle

Innerhalb der Softwareentwicklung wird oft von Vorgehens- und Prozessmodellen gesprochen. Die meisten Autoren verwenden diese Begriffe dabei als Synonyme, obwohl sich beide durchaus unterscheiden. Sommerville [Som07] definiert diese Begriffe folgendermaßen:

Das Modell eines **Softwareprozesses** beschreibt einen Satz von Tätigkeiten und damit zusammenhängenden Ergebnissen, deren Ziel die Entwicklung oder Weiterentwicklung von Software ist.

Ein **Vorgehensmodell** ist dagegen eine vereinfachte Darstellung eines Softwareprozesses, aus einer bestimmten Perspektive gesehen. Es liegt in der Natur eines Modells, dass es vereinfacht ist, und somit handelt es sich bei einem Vorgehensmodell um eine Abstraktion des tatsächlichen Prozesses, der beschrieben wird.

Eine konkretere Abtrennung der Begriffe, wie sie auch für diese Arbeit übernommen wird, ist in [LL07] beschrieben. Dabei werden **Vorgehensmodelle** als ein geeignetes Werkzeug aufgefasst, um Projektleitern und Entwicklern Hinweise zu geben, welche Tätigkeiten als nächstes auszuführen sind. Sie machen allerdings keine Aussagen über die strukturelle, technische und personelle Organisation. Werden diese Aspekte des Ablaufs, die verallgemeinert als Organisationsstruktur bezeichnet werden, in die Modellierung hinzugenommen, entsteht aus dem Vorgehensmodell ein **Prozessmodell**. Zu einer vollständigen Spezifikation eines Prozessmodells gehören außerdem die Beschreibung der Qualitätssicherung, der Konfigurationsverwaltung und des Projektmanagements. Das Vorgehensmodell an sich bildet so nur den Kern des Prozessmodells. Bekannte Vorgehensmodelle sind z.B. das Wasserfallmodell [Roy70], das Spiralmodell [Boe88] oder die komponentenbasierte Entwicklung (z.B. [CD01], [DW98]). Bekannte Prozessmodelle sind der Rational Unified Process (RUP, [Kru03]), das V-Modell XT [KBS09] oder agile Ansätze, wie Extreme Programming [Bec00].

### 2.3.2 Makro- und Mikroprozesse

Betrachtet man sowohl Vorgehens- als auch Prozessmodelle genauer, erkennt man, dass sich der Informationsgehalt von Modell zu Modell stark unterscheiden kann und demzufolge nicht nur ein einziges Vorgehens- bzw. Prozessmodell innerhalb eines Projektlebenszyklus existiert. Es entstehen bei der Entwicklung der Modelle vielmehr eine ganze Kette von Abstufungen. Beispielhaft seien hier verschiedene Abstraktionsstufen von Prozessmodellen eines Projekts auf Grundlage einer MDA-basierten Entwicklung genannt:

- Erstellung des Prozessmodells durch Anpassung eines Vorgehensmodells an die Organisationsstruktur innerhalb eines konkreten Entwicklungsprojekts.
- Bereitstellung eines Ablaufmodells der Entwicklung innerhalb einer Domänenarchitektur eines MDA-Prozesses (Model Driven Architecture-Prozess)). Dies beinhaltet z.B. die Definition der konkreten Reihenfolge der Ansteuerung der einzelnen

MDA-Werkzeugkomponenten, wie die Transformation der plattformunabhängigen Modelle (PIM) in plattformspezifische Modelle (PSM), Codegenerierung etc.

- Bereitstellung konkreter Aufgabenlisten für einzelne Mitarbeiter innerhalb eines Projektabschnitts.

Dieses Beispiel verdeutlicht eine weitere notwendige Begriffsdifferenzierung in Makro- und Mikroprozesse, wie sie in verschiedenen Quellen auf ähnliche Weise definiert wird ([Roy98], [ZGK04], [Bal08]) und [Ost05].

**Makroprozesse** beschreiben die Kontrolle und Steuerung eines Projekts über einen Entwicklungsprozess. Makroprozesse sind unabhängig bezüglich der konkreten Umsetzung von Methoden. Sie definieren grobe Abschnitte einer Projektplanung. Beispiele dafür sind Vorgehensmodelle sowie Phasen, Iterations- und Releaseplanungen.

**Mikroprozesse** dagegen beinhalten die Steuerung der täglichen Entwicklungstätigkeit. Um diesen Zweck zu erfüllen sind die Methoden durch konkrete Verfahren unterstützt, das heißt sie schreiben z.B. konkret vor, dass Java als Programmiersprache einzusetzen ist. Um die tägliche Arbeit des Entwicklers zu unterstützen und zu steuern definieren sie konkrete Aktivitäten und Arbeitsschritte.

Die Unterscheidung führt zu zwei weiteren Definitionen, die für das Verständnis dieser Arbeit notwendig sind. Ein wesentliches Unterscheidungsmerkmal zwischen Makro- und Mikroprozessen ist die Bereitstellung von Methoden im Makro- und deren konkreten Umsetzungen innerhalb des Mikroprozesses. Deshalb werden die Begriffe Methode und Verfahren der Softwaretechnik in Anlehnung an [Bal00] eingeführt.

**Methoden (Methods)** sind planmäßig angewendete und begründete Vorgehensweisen zur Erreichung von festgelegten Zielen. Methoden sind konkret im Sinne einer Weiterleitung und damit einer Aufteilung in Arbeitsschritte. Methoden sind anwendungsneutral. Sie können durch mehrere alternative oder sich ergänzende Verfahren umgesetzt werden.

**Verfahren (Techniques)** sind ausführbare Vorschriften oder Anweisungen zum gezielten Einsatz von Methoden. Sie beschreiben konkrete Wege zur Lösung bestimmter Probleme und Problemklassen. Ein Verfahren ist wegen seiner Beschränkung stärker einsatzbezogen als eine Methode.

Die Beziehung zwischen Methode und Verfahren ist in dieser Arbeit aus implementierungstechnischer Sichtweise motiviert. Methoden werden wie Muster (äquivalent zu Design Patterns [GHJV94]) eingesetzt. Verfahren gelten dann als Implementierungen bzw. Umsetzungen der Methoden. Für den Begriff Verfahren wird äquivalent der Begriff *Technik* verwendet.

### 2.3.3 Metamodelle, Modelle und Instanzen

Die bisher besprochenen Differenzierungen von Entwicklungsprozessen können benutzt werden, um Modelle anhand des Lebenszyklus eines Projekts, angefangen von einem groben Vorgehensmodell bis hin zu konkreten Aktivitätsbeschreibungen der täglichen Arbeitsplanung in Form von Aufgabenlisten, zu unterscheiden. Im Folgenden wird eine andere Dimension besprochen, die Modellierungsdimension.

Meist wird der Begriff des Prozesses verwendet, um den Ablauf bestimmter Aktivitäten in einer bestimmten Reihenfolge zu beschreiben. Jedoch stecken zwei unterschiedliche Sichten auf eine Ablaufplanung in diesem Begriff, die Modell- und die Instanzsicht [LL07]:

1. Die **Modell-Sicht** beschreibt eine abstrakte Folge von Schritten, die beliebig vielen Projekten zugrunde liegt. Man befindet sich mit dieser Sichtweise auf der Ebene eines Modells, das als Muster für ein mögliches Projekt dient.
2. Die **Instanz-Sicht** betrachtet dagegen eine konkrete und damit real ausgeführte Folge von Schritten, welche ein bestimmtes konkretes Ergebnis zur Folge hat. Diese Sichtweise betrachtet die Ebene eines konkreten Projekts und liegt damit in implementierungstechnischer Sprechweise auf der Instanzebene.

Zwischen der projektspezifischen Modellinstanz und dem projektunabhängigem Vorgehensmodell existiert demnach eine **instance-of** - Beziehung, wie sie von objektorientierten Sprachen bekannt ist. Äquivalent zu dem aufgeführten Beispiel von Prozessmodell und konkretem Projekt, befindet sich das Vorgehensmodell, welches nur einen Teil des Prozessmodells beschreibt, auf der Modellebene. Instanziert man dieses innerhalb eines konkreten Projekts erhält man den konkreten Ablaufplan für dieses Projekt.

Führt man entsprechend der Welt der Softwaremodellierung oberhalb der Modellebene noch eine Abstraktionsebene ein, erhält man die Ebene eines Prozessmetamodells. Auf dieser Ebene werden die nötigen Sprachmittel spezifiziert, mit denen es möglich wird auf der Modellebene Vorgehens- und Prozessmodelle zu beschreiben.

### 2.3.4 Die Prozessbegriffswelt des PROKRIS-Frameworks

Abbildung 2.5 zeigt einen Überblick über die definierten Begriffe. Diese Struktur bildet die Grundlage für das Prozess-Framework PROKRIS.

Die Modellierungsebenen von Softwareentwicklungsprozessen unterscheiden sich in die Metamodell-, die Modell- und die Instanzebene. Die jeweils höhere stellt dabei die Sprachmittel für die jeweils niedere zur Verfügung. Diese Einordnung ist vergleichbar mit den Modellierungsebenen der Meta Object Facility (MOF, [Obj06]). Dementsprechend werden die Ebenen auch im hier besprochenen Kontext mit M0, M1 und M2 bezeichnet.

Außerdem verdeutlicht die Abbildung, dass zu einem Prozessmodell nicht nur ein Vorgehensmodell sondern auch die Organisationsstruktur, die Qualitätssicherung, das Projektmanagement und die Konfigurationsverwaltung gehört. Dem Prozessmodell auf der Modellebene entspricht das Projekt (inkl. Organisation) auf der Instanzebene. Genauso verhält es sich mit Vorgehensmodell und Projektablauf.

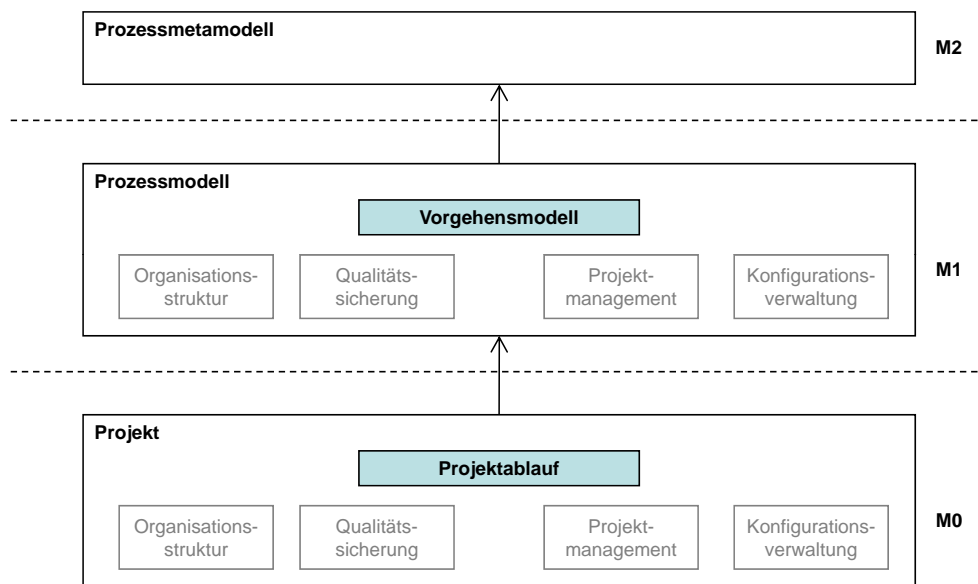


Abbildung 2.5: Überblick über die Grundbegriffe von Entwicklungsprozessen

Nicht dargestellt ist die Unterscheidung in Makro- und Mikroprozesse. In der vorliegenden Arbeit liegt diese auf einer dritten (hier nicht dargestellten) Achse der Abbildung. Vorgehensmodelle und damit auch Prozessmodelle können einerseits auf der Methodenebene und damit als grober Überblick spezifiziert werden. Dabei handelt es sich um das Modell eines Makroprozesses. Andererseits kann die Spezifikation auch auf der Verfahrensebene erfolgen. Dabei werden detaillierte Prozesse auf der Mikroebene entwickelt. Das bedeutet unter anderem, dass die generisch beschriebenen Methoden des Makroprozesses durch konkrete Verfahren und Techniken ersetzt werden. Damit wird das Prozessmodell für die Unterstützung und Steuerung der täglichen Entwicklerarbeit einsetzbar.

### 2.3.5 Software-Qualitätssicherung

Da sich die Arbeit mit der Integration von Maßnahmen zur Umsetzung nicht-funktionaler Anforderungen in Softwareentwicklungsprozessen beschäftigt, ist auch der Begriff der Software-Qualitätssicherung einzuordnen. Unter diesem Begriff werden sowohl Maßnahmen zur Sicherstellung der Produktqualität als auch der Prozessqualität zusammengefasst. Produktorientiertes Qualitätsmanagement bezieht sich darauf, Software und deren Teilprodukte auf vorher festgelegte Qualitätseigenschaften zu überprüfen. Prozessorientiertes Qualitätsmanagement bezieht sich dagegen auf den Entwicklungsprozess einer Software. Wichtige Standards sind in diesem Zusammenhang ISO 9000/9001 und das Capability Maturity Model (CMMI) [Kne07].

Einen Überblick über die Einordnung der Qualitätssicherungsmaßnahmen mit jeweils drei Beispielen gibt Abbildung 2.6. Im Kontext des PROKRIS-Frameworks bestimmt die

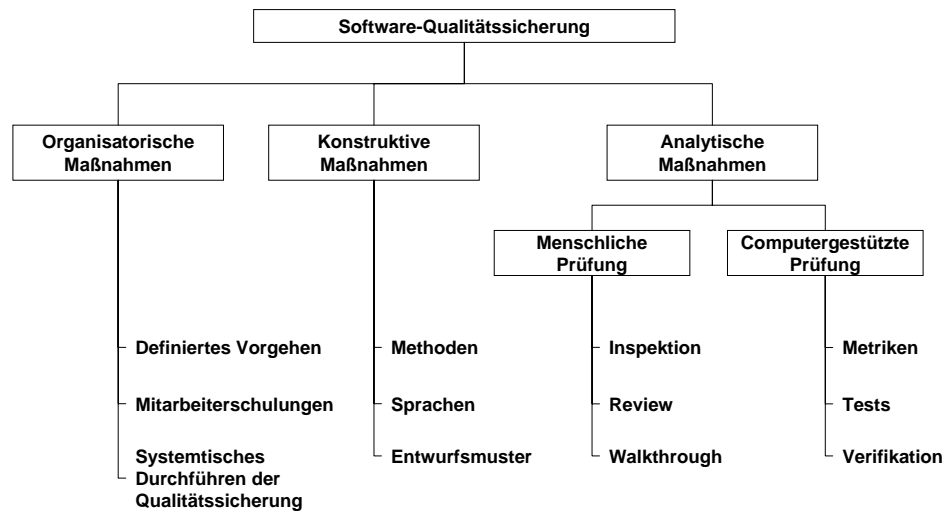


Abbildung 2.6: Gliederung der Software-Qualitätssicherung

Prozessqualität die Güte des entwickelten Vorgehensmodells zur Entwicklung laufzeitkritischer Software. Die Maßnahmen zur Produktqualität müssen dagegen in das Vorgehensmodell an sich integriert werden. Eine ausführlichere Betrachtung dazu wird in Kapitel 4 geführt. Das produktorientierte Qualitätsmanagement lässt sich in konstruktive Maßnahmen (Methoden, Sprachen, Architekturen, Checklisten) und analytische Maßnahmen (Metriken, Tests) unterscheiden. Konstruktive Eigenschaften sorgen also a priori für bestimmte Eigenschaften eines Systems. Analytische Maßnahmen ermöglichen dagegen eine Bewertung bzw. Prüfung von Qualitätseigenschaften.

In klassischen Prozessmodellen handelt es sich bei der Qualitätssicherung um einen sogenannten Querschnittsprozess innerhalb der Softwareerstellung. Als Querschnittsprozess werden unterstützende Prozesse des eigentlichen wertschöpfenden Prozesses der Softwareentwicklung bezeichnet. Entsprechend erfolgt auch die Einordnung der Qualitätssicherung in Abbildung 2.5

### 2.3.6 Einordnung in das Projektmanagement

Für das Management von Softwareprojekten gibt es verschiedene Ansätze bezüglich des Projektablaufes, die sich im Detail deutlich unterscheiden können. In der groben Ablaufstruktur und den angewendeten Managementverfahren gibt es allerdings einen Konsens. Dieser soll kurz diskutiert werden, da sich die vorliegende Arbeit daran orientiert. Diese grobe Darstellung ist für das Verständnis des PROKRIS-Frameworks und dessen Einordnung in das Gesamtbild ausreichend.

Das PMI (Project Management Institute, [PMI04]) unterteilt das Softwareprojektmanagement in fünf Prozessgruppen. Diese sind in Abbildung 2.7 dargestellt. Den Prozess-

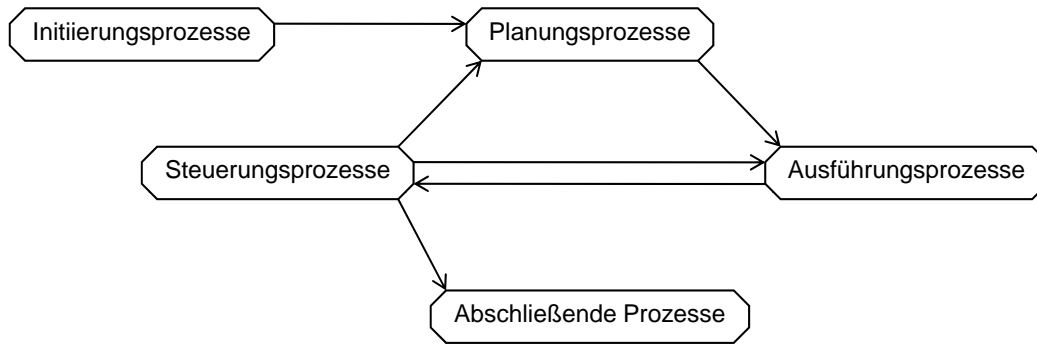


Abbildung 2.7: Prozessgruppenübersicht nach PMBook [PMI04]

gruppen sind folgende Aufgaben zugeordnet:

**Initiierungsprozesse:** Freigeben eines Prozesses oder einer Phase

**Planungsprozesse:** Festlegen und verfeinern von Zielen, Auswahl von Handlungsalternativen

**Ausführungsprozesse:** Koordinieren von Personal und anderen Einsatzmitteln

**Steuerungsprozesse:** Sicherstellen, dass Projektziele erreicht werden, Fortschrittsmessung, Planabweichungsermittlung, Einleiten von Korrekturmaßnahmen

**Abschließende Prozesse:** Formelle Abnahme eines Projekts oder einer Phase, ordnungsgemäßer Abschluss

Diesen Prozessgruppen sind Aktivitäten und Dokumente aus verschiedenen Wissensgebieten des Projektmanagements zugeordnet. Die Wissensgebiete sind unterteilt in: Integrationsmanagement, Inhalts- und Umfangsmanagement, Terminmanagement, Kostenmanagement, Qualitätsmanagement, Personalmanagement, Kommunikationsmanagement, Risikomanagement und Beschaffungsmanagement.

Diese Arbeit verwendet einen daraus abgeleiteten vereinfachten Planungsablauf zur Entwicklung eines Softwareprojekts, welcher in Abbildung 2.8 dargestellt ist. Das PROKRIS-Framework kommt dabei vorwiegend in den Prozessgruppen Initiierung, Planung und Ausführung zum Einsatz. Den einzelnen Prozessgruppen sind die während der Planung entstehenden Dokumente und deren Abhängigkeiten zugeordnet.

Im Kontext dieser Arbeit ist es wichtig den Begriff Ressource zu definieren.

Unter **Ressourcen** in der Projektabwicklung werden Personal- und Betriebsmittel verstanden, die für die Durchführung von Projektvorgängen bzw. für die Erledigung von Arbeitspaketen notwendig sind (nach [Jen08]). Konkret werden darunter in dieser Arbeit Personen, Werkzeuge und Daten verstanden.

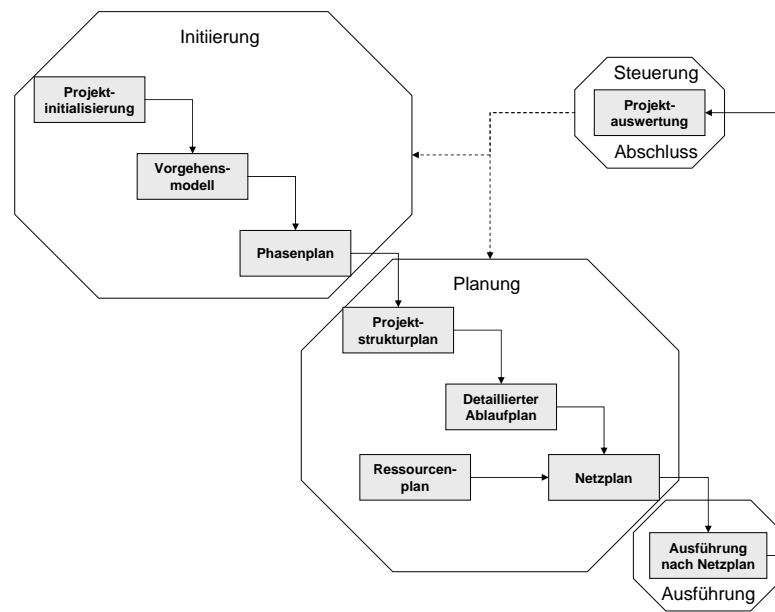


Abbildung 2.8: Planungsablauf von Softwareprojekten

## 2.4 Prozessbewertung

Zur Durchführung von Bewertungen (Assessments) von Unternehmensprozessen mit Schwerpunkt auf der Softwareentwicklung und damit auch der Verbesserung von Prozessen existieren mehrere Standards. Die bekanntesten sind CMMI [Sof] und ISO/IEC 15504 (SPICE, [ISO]). Kernpunkte dieser Standards sind die Verbesserung von Prozessen der eigenen Organisation (Process Improvement) und die Bestimmung der Prozessfähigkeit von Lieferanten (Capability Determination). Die Prozessbewertungsmethoden verfolgen im Grunde ähnliche Ansätze. Es werden verschiedene Reifegrade festgelegt, denen verschiedene Prozessgebiete zugeordnet sind. Das ist der interessante Teil für diese Arbeit, da aus dieser Zuordnung sowohl notwendige Eigenschaften des PROKRIS-Frameworks als auch von Vorgehensmodellen zur Umsetzung laufzeitkritischer Systemeigenschaften gefolgert werden können. Beispielhaft wird daher CMMI kurz erläutert. Für die weiterführende Literatur wird [Kne07] für CMMI, [HDHM06] für SPICE und [MHDZ07] für Automotive SPICE, den angepassten Standard für die Automobilindustrie, empfohlen. Alle drei Werke enthalten ebenso Verweise auf weitere Standards, sowie vergleichende Betrachtungen der verschiedenen Ansätze.

### 2.4.1 CMMI

Das Capability Maturity Model Integration (CMMI) ist ein Prozessmodell zur Beurteilung des Reifegrades von Entwicklungsprozessen. CMMI definiert Anforderungen an eine



Entwicklung mit dem Ziel eine kontinuierliche Prozessverbesserung zu unterstützen. Mit den definierten Methoden lassen sich die Stärken und Schwächen einer Organisation objektiv analysieren, Verbesserungsmaßnahmen bestimmen und diese in eine sinnvolle Reihenfolge ihrer Umsetzung bringen.

Gegenüber anderen spezialisierten Prozessmodellen für Entwicklungsorganisationen hat CMMI den Vorteil, dass es unterschiedliche Sichten auf die Organisation zusammenführt. Es spricht Projektmanagement, Entwicklung, organisatorische Unterstützung, Prozessverbesserung und Managementaufgaben in einem gemeinsamen Modell an. Es unterteilt dazu einen Prozess in verschiedene Prozessgebiete. Ein Prozessgebiet ist eine Zusammenfassung aller Anforderungen zu einem der genannten Gebiete [Kne07]. Jedes Prozessgebiet definiert sich durch Ziele, die erreicht werden sollen. Jedem Ziel wiederum sind eine oder mehrere Praktiken zugeordnet, die detailliert beschreiben, wie das Ziel erreicht werden soll. CMMI kann als stufenförmiges oder als kontinuierliches Modell eingesetzt werden.

Im stufenförmigen Modell werden die Prozessgebiete mit Hilfe der Reifegrade auf fünf Stufen eingeordnet, die jeweils für ein Entwicklungsplateau in einer Organisation stehen. Die Reifegrade sind folgendermaßen definiert (entnommen aus [Kne07]):

- 1 - Initial** Die Prozesse sind als ad hoc oder sogar chaotisch charakterisiert. Prozesse sind wenig oder nicht definiert und der Erfolg eines Projektes hängt in erster Linie vom Einsatz und der Kompetenz einzelner Mitarbeiter ab.
- 2 - Managed** fordert, die wesentlichen Managementprozesse zu etablieren, um Kosten, Zeitplan und Funktionalität von Projekten zu planen und zu steuern. Dahinter steckt die Erfahrung, dass ein funktionierendes Projektmanagement die Grundlage aller weiteren Verbesserung ist, bzw. umgekehrt ohne Projektmanagement auch alles Weitere wie z.B. definierte Prozesse nicht erfolgreich umgesetzt werden kann.
- 3 - Defined** Hier verlagert sich der Schwerpunkt der Arbeit von den einzelnen Projekten auf die Organisation als Ganzes und von den Management-Aktivitäten zu den Entwicklungsaktivitäten. Die meisten Anforderungen dieses Reifegrads beziehen sich darauf, einheitliche Prozesse für die gesamte Organisation einzuführen, während auf Reifegrad 2 noch jedes Projekt weitgehend eigene, individuelle Prozesse nutzen konnte. Dabei geht es in erster Linie um die Entwicklungsprozesse im Lebenszyklus von Systemen oder Software.
- 4 - Quantitatively Managed** Wenn eine Organisation einheitliche Prozesse eingeführt hat, dann empfiehlt das CMMI als nächsten Schritt die intensive Nutzung von Metriken und Kennzahlen, um eine bessere Entscheidungsgrundlage für Verbesserungsaktivitäten zu bekommen. Auf den niedrigeren Reifegraden werden zwar schon Metriken verwendet, vor allem zur Projektsteuerung, aber den vollen Nutzen kann man erst daraus ziehen, wenn man auf Reifegrad 3 einheitliche Prozesse eingeführt hat und unterschiedliche Kennzahlen nicht mehr auf unterschiedliche Prozesse oder gar Messmethoden zurückzuführen sind.

	Reifegrad	Prozessgebiete
5	Optimizing	Organisationsweite Innovation und Verbreitung; Ursachenanalyse und Problemlösung
4	Quantitatively Managed	Quantitatives Projektmanagement; Performanz der organisationsweiten Prozesse
3	Defined	Anforderungsentwicklung; Technische Umsetzung; Produktintegration; Verifikation; Validation; Organisationsweiter Prozessfokus; Organisationsweite Prozessdefinition; Organisationsweites Training; Integriertes Projektmanagement; Risikomanagement; Entscheidungsanalyse und -findung;
2	Managed	Anforderungsmanagement; Projektplanung; Projektverfolgung und -steuerung; Management von Lieferantenvereinbarungen; Messung und Analyse; Qualitätssicherung von Prozessen und Produkten; Konfigurationsmanagement
1	Initial	

Tabelle 2.2: Prozessgebiete des CMMI für System- und Software und ihre Zuordnung zu Reifegraden

**5 - Optimizing** Reifegrad 5 legt das Hauptaugenmerk auf die kontinuierliche Verbesserung mit der systematischen Auswahl und Einführung von Verbesserungen sowie der systematischen Analyse von auftretenden Fehlern und Problemen.

Die Zuordnung der einzelnen Prozessgebiete zu den Reifegraden ist in Tabelle 2.2 dargestellt.

In der kontinuierlichen Darstellung besitzt eine Organisation im Gegensatz zum stufenförmigen Modell keinen einheitlichen Reifegrad. Hier wird jedes Prozessgebiet getrennt nach seinem Fähigkeitsgrad (Capability Level, im Gegensatz zum Reifegrad (Maturity Level)) bewertet. In der Summe erhält man ein sogenanntes Fähigkeitsprofil, das eine wesentlich detailliertere Beschreibung ermöglicht.

#### 2.4.2 Schlussfolgerung für laufzeitkritische Systeme

Für laufzeitkritische Systeme sind folgende CMMI-Prozessgebiete aus Tabelle 2.2 besonders relevant:

**Reifegrad 2:** Qualitätssicherung von Prozessen und Produkten - Thema der Qualitätssicherung nach CMMI ist die Prüfung, dass Vorgaben für Prozesse und Arbeitsgebiete eingehalten werden. Gefordert wird die formale Korrektheit, wie die Einhaltung definierter Prozesse.

**Reifegrad 3:** Verifikation - Die Anforderungen des CMMI zur Verifikation sind relativ allgemein formuliert. Gefordert wird die Vorbereitung der Verifikation, die Auswahl der zu verifizierenden Arbeitsergebnisse sowie allgemein deren Durchführung.

Die Umsetzung dieser Anforderungen in der systematischen Prozessunterstützung von laufzeitkritischen Anforderungen in der Softwareentwicklung gestaltet sich durch die Vielfältigkeit der nicht-funktionalen Laufzeiteigenschaften als schwierig. Sowohl die Definition von Prozessen als auch die Integration von konkreten Verifikationstechniken kann entweder nur allgemeingültig und damit abstrakt erfolgen oder angepasst an konkrete NFE. Im letzteren Fall sind die Prozessmodelle nur eingeschränkt wiederverwendbar.

## 2.5 Bekannte Vorgehens- und Prozessmodelle

Zur systematischen und strukturierten Entwicklung von Software wurden eine Vielzahl unterschiedlicher Softwareentwicklungsprozesse definiert. Diese reichen von leichtgewichtigen Prozessmodellen der agilen Softwareentwicklung bis hin zu schwergewichtigen, wie dem V-Modell XT. An dieser Stelle wird sich auf eine kurze Erklärung einiger bekannter Vorgehens- und Prozessmodelle beschränkt, um den Einstieg in das Themengebiet zu erleichtern. Kapitel 4 analysiert darauf aufbauend die Unterstützung der Umsetzung nicht-funktionaler Laufzeiteigenschaften durch die Prozessmodelle.

### 2.5.1 Vorgehensmodelle

#### Wasserfallmodell

Das Wasserfallmodell war das erste veröffentlichte Vorgehensmodell für die Softwareentwicklung [Roy70]. Es gliedert das Vorgehen in die Arbeitsschritte Anforderungen, Analyse, Entwurf, Implementierung, Test und Betrieb. Es lässt außerdem eine Rückkopplung zum direkten Vorgänger zu. Bedingung ist, dass ein Arbeitsschritt erst dann abgeschlossen ist, wenn alle dafür vorgesehen Produkte fertiggestellt sind. Das Wasserfallmodell ist gut für Projekte geeignet, in denen die Arbeitsschritte deutlich voneinander trennbar und alle Risiken bereits vor Projektstart analysierbar sind [ZGK04].

#### Spiralmodell

Beim Spiralmodell [Boe88] wird das Vorgehen nicht als eine Folge von Aktivitäten mit Rückwärtsbezügen zwischen den Aktivitäten dargestellt, sondern ist als Spirale spezifiziert. Wie Abbildung 2.9 zeigt, ist das Vorgehen in vier Phasen aufgeteilt. In der ersten Phase der Zielbestimmung werden die genauen Ziele und Produkte eines Durchlaufs bestimmt. Für die definierten Ziele und deren Alternativen werden in einer zweiten Phase Risikoanalysen durchgeführt und für die gefundenen Risiken Lösungsstrategien entwickelt. In der dritten Phase erfolgt die eigentliche Produktentwicklung. In der letzten Phase wird basierend auf Reviews der nächste Zyklus geplant. Die Systementwicklung (Phase drei) kann dabei durch ein beliebiges eingebettetes Prozessmodell strukturiert werden. Da das Vorgehen durch eine Risikoanalyse gesteuert wird, kann es für sehr große und komplexe Projekte eingesetzt werden [ZGK04].

Im Spiralmodell ist außerdem das *inkrementelle* Vorgehensmodell integriert. Dabei wird ein Softwareprodukt in verschiedenen Ausbaustufen entwickelt.

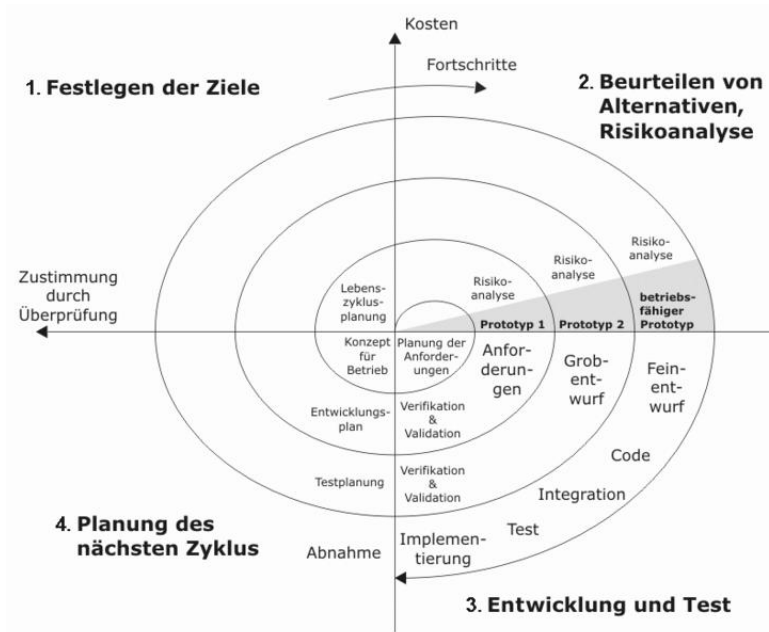


Abbildung 2.9: Spiralmodell

### 2.5.2 Prozessmodelle

#### V-Modell XT

Das V-Modell XT [KBS09, HH08] ist ein Referenzmodell zum systematischen Planen und Durchführen von komplexen und umfangreichen Entwicklungsprojekten unter Berücksichtigung des gesamten Systemlebenszyklus. Das V-Modell XT ist der Entwicklungsstandard für IT-Systeme des Bundes. Es deckt nicht nur die technische Systementwicklung ab sondern auch die Integration projektbegleitender Tätigkeiten, insbesondere der Qualitätssicherung, Konfigurationsverwaltung und des Projektmanagements.

Den Kern bildet ein logisch verknüpftes Netz von Vorgehensbausteinen. Diese enthalten Aktivitäten, Produkte und Rollen. Die Vorgehensbausteine sind die zentrale Einheit der projektspezifischen Anpassung des Referenzmodells an ein konkretes IT-Projekt. Dieser Schritt wird als *Tailoring* bezeichnet. Das Tailoring erfolgt dabei anhand verschiedener Projekttypen und Projektdurchführungsstrategien. Es werden drei Projektdurchführungsstrategien (inkrementelle, komponentenbasierte und agile Entwicklung) unterschieden. Das konkrete Vorgehensmodell wird in Projektabschnitte mit definierten Meilensteinen unterteilt. Diese werden als Entscheidungspunkte bezeichnet. Abbildung 2.10 gibt einen groben Überblick über diese Konzepte des V-Modells XT. Detaillierte Beschreibungen des Modells sind der angeführten Literatur zu entnehmen.

Das V-Modell XT definiert ein umfassendes, anpassbares Modell für Entwicklungsprojekte mit Vorlagen für Dokumente und Werkzeugunterstützung, dessen Einsatz sich für umfangreiche und kostenintensive Projekte lohnt [Bal08].

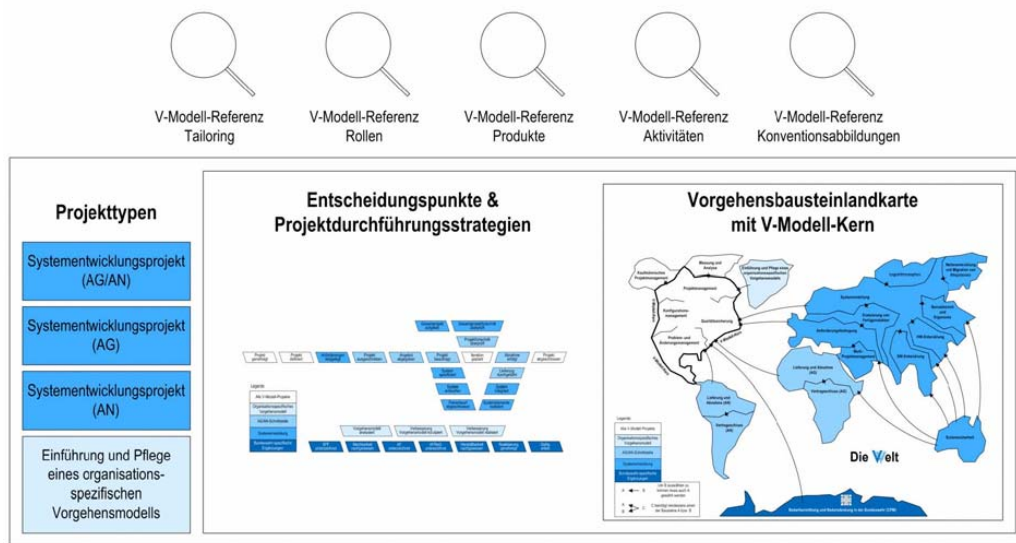


Abbildung 2.10: V-Modell XT

### Rational Unified Process

Der Rational Unified Process (RUP, [Kru03]) ist ein im Kontext der Modellierungssprache UML entwickeltes Vorgehensmodell für die Anwendungsentwicklung. Dabei ist er selbst mittels UML beschrieben. Das RUP-Vorgehen definiert ausschließlich die Entwicklung von Software und basiert auf der Integration sechs sogenannter Best-Practices<sup>2</sup>:

- Iterative Softwareentwicklung inklusive der Modellierung der Geschäftsprozesse
- Integriertes Anforderungsmanagement
- Verwendung komponentenbasierter Architekturen
- Visuelle Software-Modellierung
- Prüfung der Software-Qualität
- Kontrolliertes Änderungsmanagement

Abbildung 2.11 zeigt die Struktur im Überblick. RUP definiert zwei Dimensionen. Die horizontale Achse repräsentiert die Zeit und zeigt Aspekte des Lebenszyklus. Dabei teilt das RUP-Modell ein Projekt in vier zeitlich geordnete Phasen auf, die in mehreren Iterationen behandelt werden können:

1. Inception Phase (Projektsetup, Konzeptualisierung): beinhaltet Festlegen der Systemgrenzen, Prozessanalyse, Anforderungsanalyse

<sup>2</sup>Unter Best Practice werden bewährte Vorgehensweisen verstanden. Da der englische Begriff im Projektmanagement üblich ist, wird er auch in der Arbeit verwendet.

## 2 Grundlagen

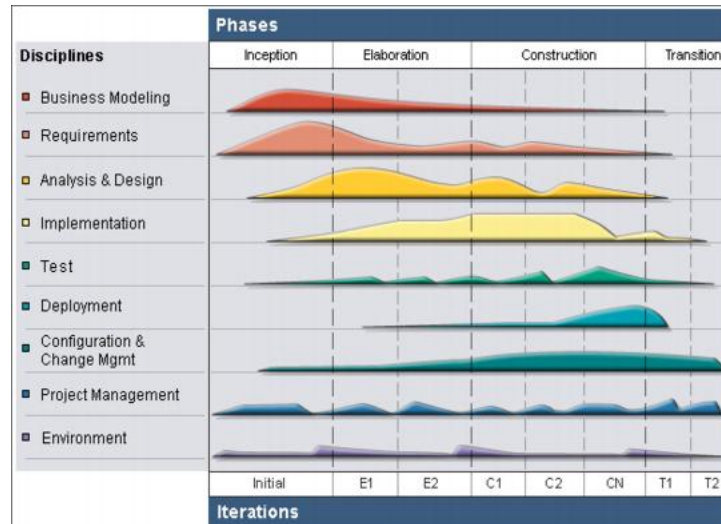


Abbildung 2.11: Rational Unified Process

2. Elaboration Phase (Ausarbeitung, Entwurf): beinhaltet Anforderungsspezifikation, Fach-, Grob- und Feinkonzept, Entwurf der Systemarchitektur, Entwurf des Testkonzepts
3. Construction Phase (Implementierung): beinhaltet Entwurf der Softwarearchitektur, Realisierung, Implementierung, Test, Abnahme
4. Transition Phase (Übertragung, Inbetriebnahme): beinhaltet Softwarepaketierung, Softwareverteilung, Inbetriebnahme

Die vertikale Dimension des RUP strukturiert die Phasen und Iterationen im RUP-Modell durch die Definition der Arbeitsschritte, welche zusammengehörige Aktivitäten logisch gruppieren. Dabei werden sechs Kernarbeitsschritte<sup>3</sup> (Core Process Workflows): Business Modeling (Geschäftsprozessmodellierung), Requirements (Anforderungsanalyse), Analysis and Design (Analyse und Design), Implementation (Implementierung), Test und Deployment (Softwareverteilung) sowie drei unterstützende Arbeitsschritte (Core Supporting Workflows): Configuration and Change Management (Konfigurations- und Änderungsmanagement), Project Management (Projektmanagement) und Environment (Entwicklungsumgebung) unterschieden.

Das RUP-Modell ist iterativ, jedoch beziehen sich die Iterationen auf Phasen oder Abschnitte im Projekt und sind eher auf Aktivitäten als auf Ergebnisse und Produkte orientiert. RUP ist eine Vorgehensweise mit eher schwergewichtiger Methodologie, vielen formalen Definitionen und Dokumenten, iterativ, architekturzentriert, Use-Case-getrieben, wohldefiniert und sehr strukturiert ist. Die konsequente und komplette Nut-

<sup>3</sup>Die Übersetzung des englischen Begriffs „Process Workflow“ in „Arbeitsschritt“ ist der deutschsprachigen Ausgabe von [Kru03] entnommen.

zung macht demnach erst bei größeren Projekten Sinn. Allerdings kann auch der RUP reduziert und angepasst werden (Tailoring).

### Agile Prozessmodelle

Agile Prozesse, wie Extreme Programming (XP, [Bec00]) oder Scrum [Sch04], repräsentieren einen völlig neuen Ansatz der Softwareentwicklung. Das „Agile Manifest“ erklärt die Werte, welche das Fundament der Agilen Softwareentwicklung bilden [Agi]<sup>4</sup>.

1. Individuen und Interaktionen gelten mehr als Prozesse und Werkzeuge. Zwar sind wohldefinierte Entwicklungsprozesse und hochentwickelte Entwicklungswerkzeuge wichtig, wesentlich wichtiger ist jedoch die Qualifikation der Mitarbeitenden und eine effiziente Kommunikation zwischen ihnen.
2. Funktionierende Programme gelten mehr als ausführliche Dokumentation. Gut geschriebene Dokumentation kann zwar hilfreich sein, das eigentliche Ziel der Entwicklung ist jedoch die fertige Software. Außerdem ist eine intuitiv verständliche Software hilfreicher als umfangreiche Dokumentation, die sowieso niemand liest.
3. Die stetige Zusammenarbeit mit dem Kunden steht über Verträgen. Ein Vertrag ist normalerweise die Grundlage für die Zusammenarbeit. Was der Kunde aber wirklich will, kann nur in ständiger Kommunikation mit ihm ermittelt werden.
4. Der Mut und die Offenheit für Änderungen steht über dem Befolgen eines festgelegten Plans. Im Verlauf eines Entwicklungsprojektes ändern sich viele Anforderungen und Randbedingungen ebenso wie das Verständnis des Problemfeldes. Das Team muss darauf schnell reagieren können.

Der Fokus der agilen Softwareentwicklung liegt darauf, Software zu bauen, die auf Nutzeranforderungen zugeschnitten ist. Außerdem soll auf veränderte Anforderungen während des Projekts unkompliziert und flexibel reagiert werden können. Die agile Softwareentwicklung hat es sich zum Ziel gesetzt, den Modellierungsaufwand, der ausschließlich für Dokumentationszwecke dient, zu minimieren.

## 2.6 Modellgetriebene Softwareentwicklung

Das Ziel der modellgetriebenen Softwareentwicklung (MDSD, [SV05]) ist die teilweise oder vollständige Generierung eines Softwaresystems aus Modellen. MDD umfasst ein weites Themenfeld. Dieser Abschnitt soll lediglich dazu dienen, die Grundidee zu erläutern. Ausgehend von formalen, abstrakten Modellen, die zunächst nur die fachliche Spezifikation eines Softwaresystems enthalten, werden durch Transformationen wiederum formale, aber konkretere Modelle erzeugt. Diese konkreteren Modelle enthalten nach der Transformation bestimmte Informationen hinsichtlich der technischen Ausführungsumgebung des zu entwickelnden Softwaresystems. Ziel des Ansatzes ist es, aus den formalen Modellen der fachlichen Spezifikation (semi)automatisch Code für verschiedene

---

<sup>4</sup>Die Übersetzung der Quelle [Agi] ist dem zugehörigen Wikipedia-Eintrag entnommen.

Architekturschichten		UML-Einordnung	UML-Beispiel
M3	Metametamodell	MOF	MetaClass
M2	Metamodell	UML	Class
M1	Modell	Klassendiagramm	Person
M0	Daten	Objektinstanz	Max

Abbildung 2.12: Modellhierarchie der Meta Object Facility (MOF)

technische Ausführungsumgebungen zu generieren. Im Kontext von MDSD existieren mehrere Standards, von denen zwei für diese Arbeit interessant sind, die Modellgetriebene Architektur (Model Driven Architecture, MDA, [Obj03]) und die Meta Object Facility (MOF, [Obj06]).

Die MDA-Spezifikation führt die Unterscheidung verschiedener Abstraktionsstufen von Systemmodellen ein [Obj03]. Diese können nach [GPR06] zusammenfassend wie folgt beschrieben werden.

**CIM - Computation Independent Model:** Das berechnungsunabhängige Modell bildet die abstrakteste Sicht auf das Systems. Es beschreibt die Anforderungen an das Gesamtsystem ohne jegliche Betrachtung der technischen Umsetzung. Es wird durch Fachbegriffe der Zieldomäne beschrieben und deswegen auch als Domain Model, oder Business Model bezeichnet. Das CIM wird als Modell während der Anforderungsanalyse eingesetzt.

**PIM - Platform Independent Model:** Das plattform-unabhängige Modell ist das abstrakteste Modell der Design-Phase. Es beschreibt die Struktur und Funktionalität des zu entwickelnden Systems, ohne auf jegliche plattformabhängige Details einzugehen.

**PSM - Platform Specific Model:** Das plattform-spezifische Modell beleuchtet die Realisierung der im PIM vorgestellten fachlichen Ansätze auf einer bestimmten Plattform. Es beschreibt dabei jedoch nicht die Art der Implementierung und beschränkt sich auf das Festlegen von Technologien.

**Code Model:** Der Programmcode wird in vielen Fällen selbst ebenfalls als Modell des Systems angesehen. Dabei handelt es sich um die konkreteste Modellsicht, in die sowohl das fachliche Wissen als auch alle Aspekte der technischen Umsetzung einfließen. Es ist das finale Ziel der Modelltransformation und stellt die lauffähige Repräsentation des Systems dar.

Die Meta Object Facility (MOF), dient der Beschreibung von Metamodellsprachen wie UML. Ein wesentliches Grundkonzept ist die Gliederung der Modellierung in vier



Modellebenen, wie in Abbildung 2.12 dargestellt. Am Beispiel von UML sollen die Schichten dieser Metamodellarchitektur kurz erläutert werden. Die oberste Ebene bildet die Metametamodellebene M3 mit MOF. Hier werden Metametaelemente (z.B. `MetaClass`) definiert. In der Metamodellebene M2 liegen die Modelle der Modellierungssprachen, z.B. das UML-Metamodell. Hier werden die Syntax und die Semantik der UML Sprach-elemente (z.B. `Class`) zum Modellieren eines Systems definiert. Die Modellebene M1 enthält die vom Benutzer modellierten UML Diagramme mit deren Diagrammelementen (z.B. der Klasse `Person`). Die unterste Ebene M0 ist die Datenebene. Diese repräsentiert Elementinstanzen zur Laufzeit, wie das Objekt `Max:Person`.

## 2.7 Wiederverwendung von Vorgehens- und Prozessspezifikationen

Nachdem bereits verschiedene Aspekte der Differenzierung von Softwareprozessen behandelt wurden, wird in diesem Abschnitt eine weitere Sichtweise auf die Prozessmodellierung diskutiert, die Wiederverwendung von Teilen einer Prozessspezifikation. Wiederverwendung spielt in der Softwareentwicklung schon seit geraumer Zeit eine immer größere Rolle. Die Vorteile sind Zeitersparnis, bessere Wartbarkeit und Verbesserung der Qualität der Softwareentwicklung.

### 2.7.1 Prozessbausteine

Bei Entwicklungsprozessen gibt es einen Richtungswechsel weg von fest vordefinierten Vorgehensbeschreibungen hin zu agileren Methoden und dem Einsatz von Best Practice Methoden. Diesem Trend können sich auch die „schergewichtigen“ Ansätze nicht entziehen. Ansätze wie der Rational Unified Prozess und das V-Modell XT basieren schon auf der Grundidee, dass Vorgehensmodelle und Projektabläufe aus vielen wiederkehrenden Aktivitäten, Rollendefinitionen und Artefakten bestehen. Diese Definitionen können ähnlich des Wiederverwendungsansatzes von Softwarekomponenten natürlich auch als wiederverwendbare Modelle oder Ablaufstrukturen behandelt werden.

Die wiederverwendbaren Artefakte der Prozessmodellierung werden auch als *Prozess- oder Vorgehensbausteine* bezeichnet. Dabei ist die Granularität der Bausteine in den klassischen Ansätzen, wie RUP oder V-Modell XT, sehr groß und die Anpassung erfolgt quasi vom Großen zum Kleinen. Das bedeutet, umfangreiche Spezifikationen von Entwicklungsprozessen werden durch *Tailoring* angepasst. Als Tailoring wird die projektspezifische Anpassung bezeichnet. So reduziert die Tailoring-Phase im V-Modell XT das Volumen auf solche Prozessanteile, die in einem bestimmten Projektkontext erforderlich sind. Projekte mit unterschiedlichem Kontext sind im V-Modell XT beispielsweise Projekte, in denen ein Softwaresystem erstellt wird oder Projekte, in denen ein System „nur“ beschafft wird. Im ersten Fall spricht das V-Modell XT von einem Auftragnehmerprojekt, im letzteren von einem Auftraggeberprojekt. In einem derartigen Ansatz ist die Integration eigener Bausteine schwierig. Andere Arbeiten ([GMP<sup>+</sup>03], [Hau06], [HS03], [Rau01]) beschreiben einen anderen Weg, sozusagen vom Kleinen zum Großen. Aus ei-

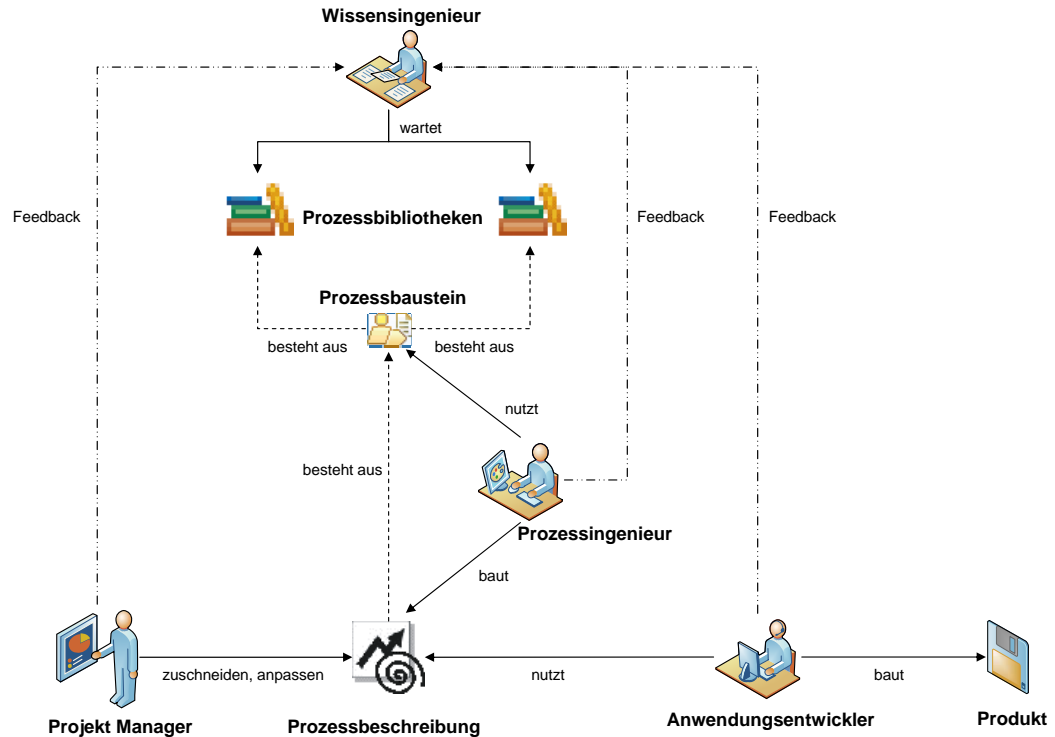


Abbildung 2.13: Einsatzszenario eines Prozess-Frameworks

ner Bibliothek wiederverwendbarer Bausteine können geeignete ausgewählt und zu einem Prozessmodell zusammengesetzt werden. Dieser Ansatz ist offensichtlich flexibler bei der Integration eigener Prozessbausteine und soll daher für das PROKRIS-Framework eingesetzt werden.

### 2.7.2 Prozess-Framework

Die definierten Prozessbausteine werden in einer *Prozessbibliothek* verwaltet. In einigen Arbeiten ([EPF], [HS03]) wird synonym der Begriff Methodenbibliothek verwendet. Da in dieser Arbeit jedoch eine Trennung in Methoden und Verfahren vorgenommen wird, wird ausschließlich der Begriff Prozessbibliothek verwendet, um eine Verwechslung zu vermeiden.

Eine oder mehrere Prozessbibliotheken bilden den Kern eines Prozess-Frameworks. Die Prozessbibliothek fungiert innerhalb des Frameworks als ein Repository für wiederverwendbare Prozesselemente, wie Aktivitäten, Artefakte und Rollen, welche eine Organisation nutzen kann, um einen Entwicklungsprozess zu konstruieren. Sie bildet somit eine strukturierte Sammlung von Prozessartefakten, welche durch zusätzliches Material (z.B. Dokumentvorlagen) unterstützend hinterlegt werden können. Abbildung 2.13 zeigt die Nutzung einer solchen Prozessbibliothek durch verschiedene Stakeholder.

Der Prozessingenieur nutzt die Bibliothek, um passende Bausteine auszuwählen und an Umgebungsbedingungen anzupassen. Darauf aufbauend kann er Prozesse konstruieren, welche durch den Projektmanager in einem abgestecktem Rahmen noch an projektspezifische Randbedingungen (z.B. durch die Planung von Meilensteinen) angepasst werden können. Entwickler nutzen die Prozessbeschreibungen als Leitfaden für ihre Arbeit. Alle Stakeholder<sup>5</sup> können Feedback über ihre Erfahrungen im Umgang mit dem bereitgestellten Prozesswissen an den Methodeningenieur geben. Dieser wertet es aus und pflegt es in die Prozessbibliothek ein.

Um die beschriebenen Arbeiten zu unterstützen benötigt ein Prozess-Framework zusätzlich zur Prozessbibliothek Autorenwerkzeuge zur Bereitstellung und Veröffentlichung von Prozessbausteinen und Prozessen sowie zum Erstellen und projektspezifischen Anpassen von Prozessen.

## 2.8 Beschreibungstechniken für Prozesse

Im Abschnitt 2.3 wurden unterschiedliche Modellierungsebenen von Prozessen beschrieben. Dabei wurden die Metamodell-, Modell- und Instanzebene unterschieden. Für jede dieser Ebenen ist es notwendig Sprachkonzepte zur Vorgehens- und Prozessmodellierung bereitzustellen. Dabei ist grundsätzlich zwischen Sprachen der Modellierungsebene (M1 und M2) und der Ausführungsebene (M0) zu unterscheiden. In der Implementierungswelt ist dies vergleichbar mit dem UML-Modell (und dessen Metamodell) und der Java-Implementierung einer Anwendung.

In diesem Kapitel werden nur die beiden in dieser Arbeit verwendeten Sprachen eingeführt. Einen Überblick über weitere Beschreibungsmittel und deren Einordnung bezüglich dieser Arbeit befindet sich im Kapitel 10

### 2.8.1 Prozessmetamodelle und Prozessmodelle

Als Metamodell bezeichnet man ein Modell, welches die Syntax eines anderen Modells definiert. Es stellt quasi eine Modellierungssprache für Modelle bereit. Da sich die Definitionen des Metamodells ausschließlich auf die Syntax und nicht den Gegenstand des anderen Modells beziehen, sind Modellelemente Instanzen von Metamodellelementen. Das Konzept der Metamodellierung ist für die Bereitstellung von wiederverwendbaren Prozessbeschreibungen von großer Bedeutung, da nur durch die Verwendung eines Metamodells eine einheitliche Sprachdefinition realisierbar ist.

Ein vielversprechenderer Ansatz für die Beschreibung der Prozessbausteine ist das Software Process Engineering Metamodel der OMG (SPEM, [Obj08a]). SPEM wurde von der OMG explizit für die Modellierung und den Austausch von Softwareentwicklungsprozessen entwickelt. Die damit mögliche Vereinheitlichung der Metamodelle unterschiedlicher Vorgehensbeschreibungen gestattet außerdem die Integration von Methoden

---

<sup>5</sup>Stakeholder sind Personen, Institutionen oder Dokumente, die von der Entwicklung und vom Betrieb eines Softwaresystems betroffen sind. Sie repräsentieren Gruppen mit gleicher Interessenlage und gleicher Sicht auf das System.

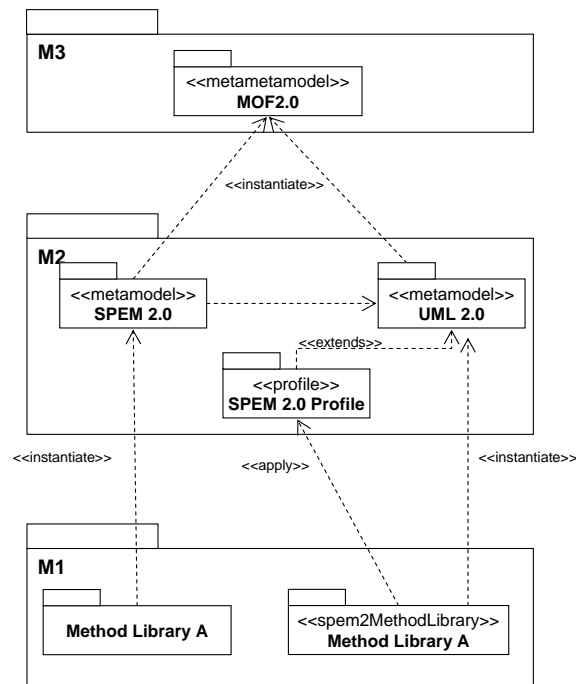


Abbildung 2.14: Modellierungsebenen von SPEM 2.0 und UML 2.0 nach [Obj08a]

und Konzepten aus verschiedenen Vorgehensmodellen. SPEM ist somit ein Standard für eine Modellierungssprache von Prozessen und Prozessfamilien.

SPEM liegt in der aktuellen Version 2.0 vor. Das SPEM 2.0 Metamodell wird durch eine Metamodellierungssprache definiert. Dazu wird die Metadata Object Facility (MOF, [Obj06]) genutzt. Wie bereits beschrieben, wird auch UML 2.0 mittels MOF modelliert. SPEM 2.0 nutzt also die gleiche Modellierungshierarchie wie UML 2.0. Abbildung 2.14 zeigt die Einordnung von UML und SPEM in die verschiedenen Modellierungsebenen. Sowohl UML als auch SPEM liegen auf der Metamodellebene M2. Diese Einordnung ermöglicht die Definition von SPEM auf der Basis einer Teilmenge von UML indem es die selben Metamodelllemente (in MOF auf M3 modelliert) wie UML verwendet. SPEM 2.0 definiert auf der M2-Ebene Konzepte wie Rollen, Aufgaben und Artefakte sowie Beziehungen zwischen diesen. Auf der M1-Ebene stellt dann beispielsweise **Method Library A** (Abb. 2.14, linke Seite) konkrete Instanzen dieser Rollen und Artefakte, wie die Rolle **Systemanalyst** bereit. Auf M0 agiert dann eine konkrete Person, z.B. Max Mustermann als Instanz der Rolle **Systemanalyst** zur Laufzeit des Prozesses (nicht dargestellt).

Ein weiteres Beispiel zeigt Abbildung 2.15 auf der linken Seite. Auf M2 ist das Metamodellelement **Artifact** definiert. Wie zu sehen, ist **Use Case** auf M1 eine direkte Instanz dieser Meta-Klasse **Artifact**, welche wiederum eine Instanz der Meta-Meta-Klasse **Class** auf der M3-Ebene ist. Eine **Use Case**-Instanz auf der M0-Ebene, wie

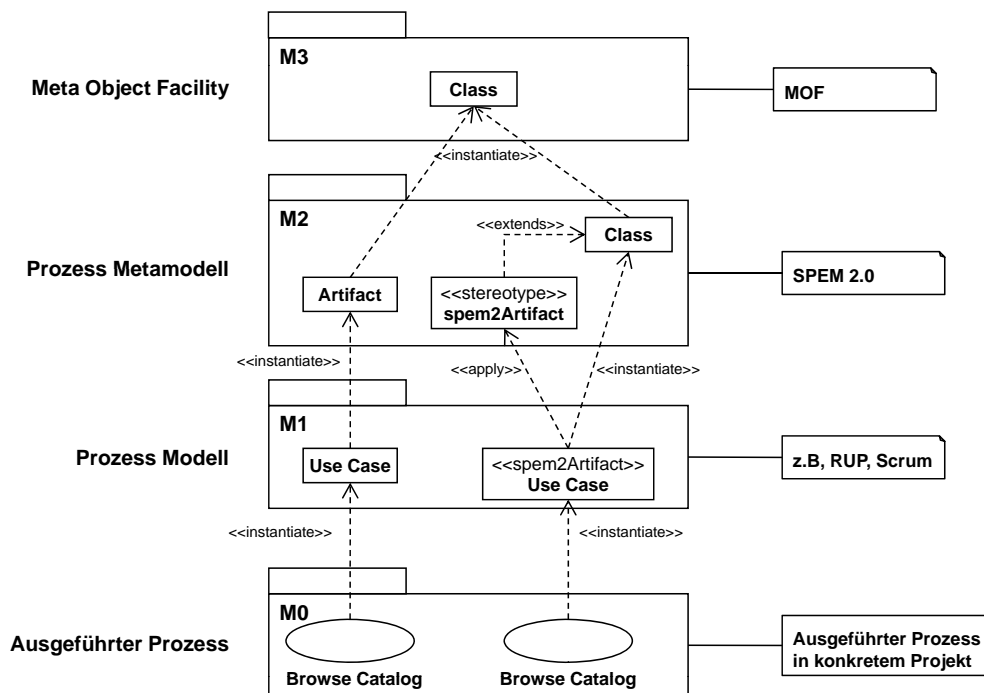


Abbildung 2.15: Beispiel für die Instanziierung der Modellschichten nach [Obj08a]

**Browse Catalog**, welche während der Entwicklung eines WWW-Shops erstellt wird, ist dann eine direkte Instanz des M1-Modellelements **Use Case**.

SPEM 2.0 stellt aber nicht nur ein Metamodell, sondern als Alternative dazu ein SPEM 2.0 Profil zur Verfügung. Dieses definiert eine Anzahl von UML Stereotypen. Der Vorteil der Profilnutzung ist, dass keine speziellen Werkzeuge zur Modellierung notwendig sind, sondern dass auf Standard-UML-Werkzeuge mit Profilunterstützung zurückgegriffen werden kann. Abbildung 2.14 ordnet auf der rechten Seite dieses SPEM 2.0 Profil in die Metaebenen von MOF ein. Die Definition der Stereotypen erfolgt auf der Metamodellebene M2 basierend auf UML-Metamodellelementen. Die Stereotypen können auf der Modellebene M1 zur Modellierung des Inhalts von Prozessbibliotheken (z.B. **Method Library A**) genutzt werden. Das Profil definiert lediglich die Präsentation der Elemente. Alle semantischen Definitionen beruhen weiterhin auf der Metamodelldefinition. Abbildung 2.15 zeigt auf der rechten Seite das Beispiel unter Nutzung des SPEM 2.0 Profils.

Zusammenfassend kann gesagt werden, dass SPEM 2.0 sowohl Konzepte für die Metamodell- als auch für die Modellebene der Vorgehens- und Prozessspezifikation bereitstellt. Es ist das derzeit am weitesten entwickelte Konzept für diesen Anwendungskontext und wird daher innerhalb dieser Arbeit genutzt. Es weist natürlich noch Defizite auf, die in den späteren Kapiteln diskutiert werden.

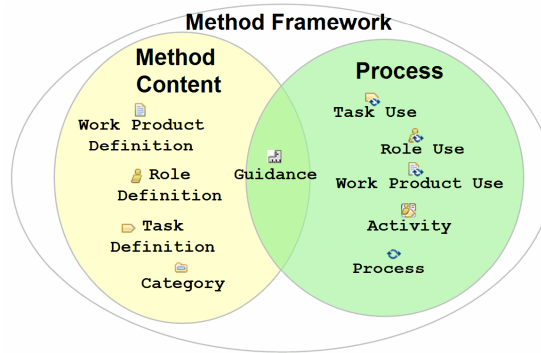


Abbildung 2.16: Überblick über die Konzepte des SPEM-Metamodells (aus [Obj08a])

### 2.8.2 Modellierung von Prozessbausteinen

SPEM 2.0 bietet eine sehr gute Ausgangsbasis für die Modellierung von Prozessbausteinen, da es konsequent die Beschreibung einzelner Methodeninhalte verfolgt, welche zur Komposition von Prozessen bereitgestellt werden können. Das ist ein Vorteil für den Einsatz innerhalb des PROKRIS-Frameworks und soll daher kurz erläutert werden.

Die strikte Trennung von Methodeninhalten und Prozessmodellen ist eine wesentliche Eigenschaft von SPEM 2.0. Durch diesen Aufbau können die Paradigmen Wiederverwendung (Reuse) und Trennung von Zuständigkeiten (Seperation of Concerns) der Objekt-orientierung auch auf Prozessmodellierungen angewendet werden. Es wird möglich, Methoden als Prozessbausteine zur Verfügung zu stellen, welche flexibel zur Modellierung von Vorgehens- und Prozessmodellen eingesetzt werden können. Im Rational Unified Process (RUP) ist diese Trennung bereits implizit umgesetzt. Die vertikale Dimension strukturiert den RUP in zusammengehörige Arbeitsschritt (Abbildung 2.11). Diese sind im Sinne von SPEM 2.0 Methoden. Die horizontale Dimension bringt die Methoden in einen zeitlichen Bezug und stellt somit aus den Methoden Prozesse zusammen.

Abbildung 2.16 [Obj08a] stellt die Metamodellelemente, wie sie in SPEM 2.0 definiert sind, und deren Zuordnung zum Methodeninhalt (**Method Content**) bzw. Prozessmodell (**Process**) dar. Methodeninhalte definieren sich dabei als eine Auswahl von bewährten Vorgehensweisen und modellieren wie bestimmte Aufgaben (**Task Definition**) ausgeführt werden. Zum Methodeninhalt gehört außerdem die Definition von Arbeitsergebnissen (**Work Product Definition**) und Rollen (**Role Definition**).

Die Idee dieser Trennung lässt sich gut am Metamodellelement **Task Definition** erläutern, welches dem Paket **Methodeninhalt** und nicht dem Paket **Prozessmodell** zugeordnet ist (Abbildung 2.16). Als Beispiel für eine Aufgabendefinition (**Task Definition**) wird **Anforderungsanalyse** eingeführt. Diese Aufgabe erfordert als Eingabe das Arbeitsergebniss (**Work Produkt Definition**) **Ausschreibung** und liefert unter anderem als Ausgabe das Arbeitsergebnis **Use Case Model**. Projekte nach dem Wasserfallmodell führen die **Anforderungsanalyse** genau einmal innerhalb der ersten Projektphase durch. Iterativ vorgehende Projekte führen diese Aufgabe dagegen mehrmals in verschiedenen Entwicklungsphasen aus und verändern bzw. erweitern dabei das Arbeitsergebnis

**Use Case Modell.** Die Definition der Aufgaben ist demnach unabhängig vom gewählten Vorgehensmodell. Daher werden die Aufgabendefinitionen auch unabhängig von einem konkreten Prozessmodell definiert. Die Strukturierung eines Prozesses erfolgt in SPEM 2.0 durch das Metamodellelement **Activity** (Aktivität). Beim Modellieren eines konkreten Prozessmodells (**Process**) werden die jeweiligen Aufgabendefinitionen innerhalb der Aktivitäten nur referenziert. Verschiedene Prozessmodelle können somit verschiedene Aktivitäten spezifizieren, aber auf die gleichen Aufgabendefinitionen verweisen. Diese Vorgehensweise gilt auch für die anderen Elemente des Methodeninhalts. Die Referenzen vom Prozessmodell auf den Methodeninhalt werden mit Hilfe des **Use**-Bezeichners repräsentiert.

Anleitungen (**Guidance**), wie Leitfäden, Whitepapers, Checklisten, Beispiele, werden am Schnittpunkt von Methodeninhalt und Prozess definiert (Abbildung 2.16). Anleitungen können so als zusätzliches Wissen sowohl für Definitionen im Methodeninhalt als auch für spezifische Prozessmodelle bereitgestellt werden.

Das Eclipse Process Framework (EPF, [EPF]) und das zugehörige Werkzeug EPF Composer stellen die derzeit am vollständigsten umgesetzte Referenzimplementierung des Standards zur Verfügung. Dies ist ein Grund, weshalb das PROKRIS-Framework auf diesem Werkzeug aufsetzt. Ein zweiter ist die Erweiterbarkeit des Werkzeugs, da es „Open Source“ ist. Außerdem sind Eclipse Projekte relativ einfach durch den Plugin-Ansatz erweiterbar.

### 2.8.3 Ausführbare Prozessmodelle

SPEM 2.0 wurde von der OMG mit dem Ziel entwickelt, ein einheitliches Modellierungsformat für Vorgehens- und Prozessmodelle bereitzustellen. Es ist dagegen nicht darauf ausgelegt Ausführungsdetails von Prozessen zu modellieren. Daher wird für das PROKRIS-Framework eine weitere Beschreibungstechnik benötigt. Es gibt eine Vielzahl von Ansätzen zur Modellierung von ausführbaren Abläufen. Kapitel 10 gibt dazu einen Überblick. An dieser Stelle werden nur die Grundkonzepte von jPDL (jBPM Process Definition Language, [jPD]) erläutert, da eine Erweiterung dieser im PROKRIS-Framework eingesetzt wird.

jPDL ist eine von jBoss entwickelte XML-basierte Prozessbeschreibungssprache. Sie beruht auf der Beschreibung von Zustandsgraphen und enthält damit Konzepte wie Knoten, Transitionen, Entscheidungen, Forks und Joins. Eine umfassende Dokumentation aller jPDL Elemente befindet sich in [jPD].

Ein Prozessmodell wird in jPDL durch einen Prozessgraphen in Form eines XML-Baums repräsentiert, welcher mit einem Startzustand beginnt und mit einem Endzustand abschließt. Diese Zustände sind spezielle Knoten, die über Transitionen mit den übrigen Zuständen des Prozesses verbunden sind. Listing 2.1 stellt eine einfache jPDL-Beschreibung des in Abbildung 2.17 dargestellten Prozesses vor.

jPDL definiert verschiedene Knotentypen, welche auf unterschiedliche Art die Prozessbearbeitung beeinflussen. Die wichtigsten sind:

**Zustand <state>:** Dieser Knoten repräsentiert einen Wartezustand. Die Prozessabar-

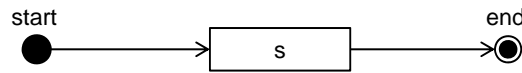


Abbildung 2.17: Beispiel eines einfachen Prozessgraphen

beitung wird beim Erreichen dieses Zustands in einen Wartezustand versetzt bis eine erneute Eingabe den Ablauf wieder anstößt.

**Aufgabenknoten <task-node>:** Im Gegensatz zum Zustand enthält der Aufgabenknoten eine Reihe von Aufgaben (tasks). Diese werden beim Betreten des Knotens instantiiert und müssen abgearbeitet werden, bevor der Prozessablauf weitergeführt werden kann.

**Entscheidungsknoten <decision>:** An diesem Knoten wird abhängig von einer getroffenen Entscheidung eine abgehende Transition und somit der weitere Ausführungspfad ausgewählt.

**Verzweigungsknoten <fork>, <join>:** Mit Hilfe der Verzweigungsknoten können parallele Ausführungspfade definiert werden.

Außerdem gibt es noch sogenannte <Super State>s, welche Knoten gruppieren können. Somit ist es möglich, hierarchische Strukturen und Phasen innerhalb einer Prozessdefinition zu modellieren.

```

1 <process-definition>
2   <start-state>
3     <transition to='s' />
4   </start-state>
5   <state name='s'>
6     <transition to='end' />
7   </state>
8   <end-state name='end' />
9 </process-definition>

```

Listing 2.1: Prozessdefinition zu Abbildung 2.17

Während der Abarbeitung des Prozessgraphens sind verschiedene Ereignisse (**events**) definiert. Diese ermöglichen es, auf bestimmte Situationen in der Abarbeitung zu reagieren. Standardereignisse in jPDL sind das Durchlaufen einer Transition und das Betreten bzw. Verlassen eines Knotens. Den Ereignissen können wiederum Aktionen (**actions**) zugeordnet werden, welche beim Auftreten des Ereignisses auszuführen sind. Die Aktionen werden in jPDL durch sogenannte **ActionHandler** in Java implementiert. Diese erhalten beim Aufruf durch eine Workflowmaschine eine Referenz auf den Ausführungskontext und können somit auf Prozessvariablen zugreifen und das Prozessverhalten beeinflussen.



## 2.9 Ausführung von Prozessmodellen

Eines der Hauptziele des PROKRIS-Frameworks ist es, die Ausführung von Prozessmodellen der Mikroebene zu unterstützen. Diese Arbeit diskutiert dafür den Einsatz einer Workflowmaschine als Grundlage für die Architektur der Prozess-Ausführungskomponente. Daher wird in diesem Kapitel die Standardarchitektur eines Workflow Management Systems (WfMS), welche von der Workflow Management Coalition (WfMC) definiert wurde, und jBPM als eine Implementierung vorgestellt.

### 2.9.1 Standardarchitektur eines Workflow Management Systeme

Unter einer Vorgangssteuerung (Workflow<sup>6</sup>) versteht die WfMC einen teilweise oder vollständig automatisierten (Geschäfts-)prozess, in dem Dokumente, Informationen oder Aufgaben zwischen Teilnehmern entsprechend einer Menge von Ausführungsregeln übertragen werden. Mittels einer Workflowbeschreibungssprache, beispielsweise der bereits vorgestellten jPDL, ist es möglich, Workflows in einer für Maschinen verständlichen formalen Form zu spezifizieren. Eine Workflowmaschine (Workflow Engine) ermöglicht es eine Workflowspezifikation zu interpretieren und auszuführen. Sie hat nach der WfMC folgende Aufgaben zu erfüllen [Wor95]:

- Bereitstellung bzw. Modellierung einer Prozessdefinition
- Erzeugung und Ausführung von Prozessinstanzen
- Bearbeitung von Aufgaben; die Information, Verwaltung und Auswertung der Aufgaben erfolgt über eine Nutzerschnittstelle
- Integration externer Anwendungen
- Überwachung, Verwaltung, Analyse (Reorganisation) eines Prozesses

Abbildung 2.18 zeigt eine schematische Darstellung eines Workflow Management Systems, wie es das Referenzmodell der WfMC definiert. Es beschreibt fünf verschiedene Schnittstellen (Interfaces), welche durch ein WfMS angeboten werden sollten, um die oben genannten Aufgaben zu erfüllen. Die drei wichtigsten sind:

**Interface 1:** Das **Process Definition** Interface dient als Import- und Exportschnittstelle von Prozessbeschreibungen. Die Definition dieser Schnittstelle ermöglicht es, die Ausführung von Prozessen von deren Definition und damit auch von der Auswahl des Werkzeugs zur Prozessmodellierung zu lösen.

**Interface 2:** Das Interface **Workflow Client Application** bildet die Schnittstelle zwischen Nutzer und Workflowmaschine. Mit Hilfe dieser Schnittstelle können die Verwaltung von anstehenden Aufgaben, das Erzeugen und Bearbeiten von Aufgaben, die Verwaltung von Prozessinstanzen und die Datenverwaltung abgewickelt werden.

---

<sup>6</sup>Diese Arbeit verwendet vorwiegend den auch im Deutschen gebräuchlichen Begriff „Workflow“.

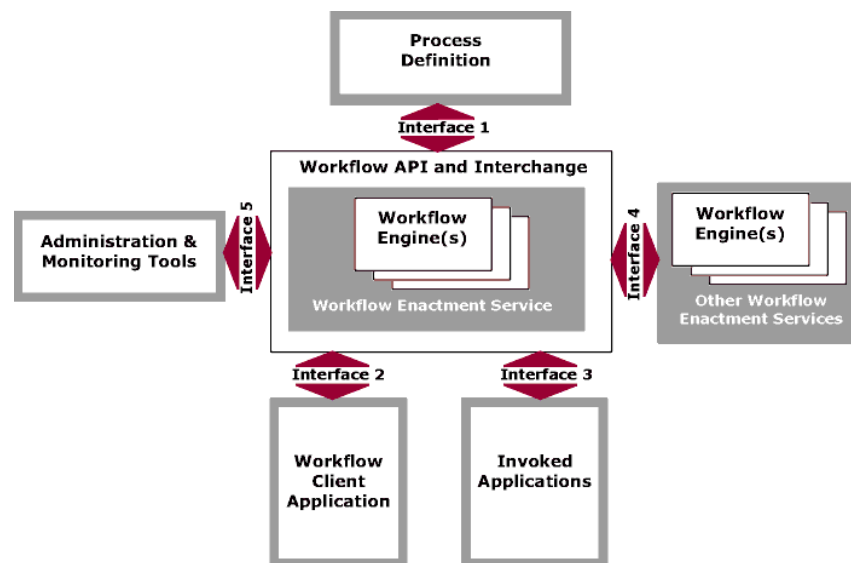


Abbildung 2.18: Workflow Referenz-Modell der WfMC (aus [Wor95])

**Interface 5:** Das Interface **Administration and Monitoring Tools** definiert die Administration und Überwachung von Workflowmaschinen mit verschiedenen Überwachungsprogrammen.

Neben diesen Basisschnittstellen, welche die oben definierten Aufgaben eines Workflow Management Systems erfüllen, definiert das Referenzmodell zwei weitere:

**Interface 3:** Das **Invoked Application** Interface bildet eine Schnittstelle zur Kommunikation zwischen Workflowmaschine und Programmen. Hierbei müssen Funktionen für die Aktivierung von Aktivitäten in der angeschlossenen Anwendung und für die Übergabe der Aktivitätsergebnisse implementiert sein.

**Interface 4:** Das Interface **Workflow Interoperability** definiert, wie Prozesse auf mehreren Workflowmaschinen gekoppelt ablaufen können. Hierbei müssen Aktivitäts- und Unterprozessaktivierungen, Prozessstatus, prozessrelevante Daten und Synchronisationsdaten übertragen werden.

### 2.9.2 Java Business Process Management System (jBPM)

Eine Implementierung der WfMC-Architektur stellt JBoss mit dem jBPM (Java Business Process Management, [jBP]) zur Verfügung. Mit diesem können sowohl jPDL- als auch BPEL-Prozesse ausgeführt werden. jBPM stellt eine einfach erweiterbare Java API dar, welche verschiedene Funktionen zur Verfügung stellt, um graphenbasierte Prozesse zu definieren und auszuführen. Da sie zudem noch „Open Source“ ist, wird sie für die prototypische Implementierung der PROKRIS-Ausführungskomponente eingesetzt.

jBPM implementiert drei Schnittstellen: Deployment, Execution und Monitoring. Diese können folgendermaßen definiert und in die WfMC-Architektur eingeordnet werden.

**Deployment:** entspricht dem Interface 1 des WfMC-Modells. Eine durch einen Prozessingenieur entworfene Prozessbeschreibung kann über diese Schnittstelle in die Maschine eingebracht werden.

**Monitoring:** entspricht dem Interface 5 des WfMC-Modells. Über diese Schnittstelle ist es möglich auf Prozessdaten zuzugreifen und Prozesse zu starten. Sie wird von zwei Nutzergruppen verwendet. Ein Prozess (Projekt) verfügt einerseits über einen Administrator oder Manager. Auf der anderen Seite sind die Nutzer des Prozesses, welche als Prozessbeteiligte Aufgaben innerhalb des Projekts zu erfüllen haben.

**Execution:** entspricht dem Interface 2 des WfMC-Modells. Die Schnittstelle dient der schrittweisen Abarbeitung des Prozesses und wird in der Regel nur von den Beteiligten des Prozesses verwendet.

Die Speicherung der Prozessdefinitionen und Prozessinstanzen wird in einer Datenbank vorgenommen. Die Kernkomponente von jBPM implementiert alle notwendigen Methoden, um Prozessdefinitionen und Prozessinstanzen zu erzeugen, zu aktualisieren und wiederherzustellen.



## 3 Die Idee des PROKRIS-Ansatzes

Ein wichtiges Ziel der Arbeit ist die Bereitstellung eines Frameworks zur systematischen Prozessunterstützung bei der Entwicklung von Softwaresystemen mit laufzeitbasierten nicht-funktionalen Eigenschaften. Dazu wurde sowohl eine neue Methodik als auch das PROKRIS-Framework zur Unterstützung der Bereitstellung und Nutzung der erweiterten Vorgehensmodelle entwickelt. Die Methodik umfasst eine Generalisierung von notwendigen Vorgehensmodellerweiterungen der Makroprozessmodelle durch Prozessmuster. Außerdem bietet sie die Möglichkeit durch eine NFE-kontextbasierte Anpassung die Makroprozesse stufenweise an ein spezifisches Projekt und die darin geforderten nicht-funktionalen Eigenschaften anzupassen.

Zur Einführung diskutiert dieses Kapitel anhand eines Beispiels die Herausforderungen, die nicht-funktionale Eigenschaften an die Prozessmodellierung stellen. Im Anschluss darin wird die grundlegende Idee des PROKRIS-Ansatzes als Lösung vorgestellt.

### 3.1 Die Herausforderungen laufzeitkritischer NFE an Vorgehensmodelle

Zum besseren Verständnis der grundlegenden Idee von PROKRIS-Methodik und -Framework wird der Lebenszyklus eines Entwicklungsprojekts für NFE-kritische Software anhand eines Beispiels aufgezeigt. Als Beispiel dient ein fiktives Entwicklungsprojekt aus der Automobilindustrie, welches auch im SuReal-Projekt als Fallstudie eingesetzt wird. Im SuReal-Projekt wird sowohl ein Entwicklungsprozess für zeitkritische Software als auch Methoden zur Modellierung und Durchsetzung derartiger Eigenschaften entwickelt.

#### 3.1.1 Lebenszyklus am Beispiel des „SuReal-Projekts“

Im gewählten Beispielprojekt soll die Software für ein neues Bremssystem einer PKW-Baureihe entwickelt werden. Das Bremssystem soll automatisch auf zu nahe kommende vorausfahrende Fahrzeuge reagieren und somit Kollisionen vermeiden. Das bedingt kritische Reaktionszeiten des Systems, welche sich in zeitbasierten nicht-funktionalen Anforderungen widerspiegeln.

Der Projektverlauf folgt den klassischen Projektzyklen, wie sie in Kapitel 2 beschrieben sind. Abbildung 3.1 bietet einen grafischen Überblick über die Projekt- und Vorgehensplanung des Beispielprojekts und deren Einflussfaktoren. In der folgenden Beschreibung wird der Fokus gezielt auf solche Maßnahmen gelegt, die zur durchgängigen Behandlung der Realzeitanforderungen notwendig sind. Im Kern der Abbildung sind die Projektplanungsphasen bzw. deren Artefakte, wie sie für das Management von Softwareprojekten

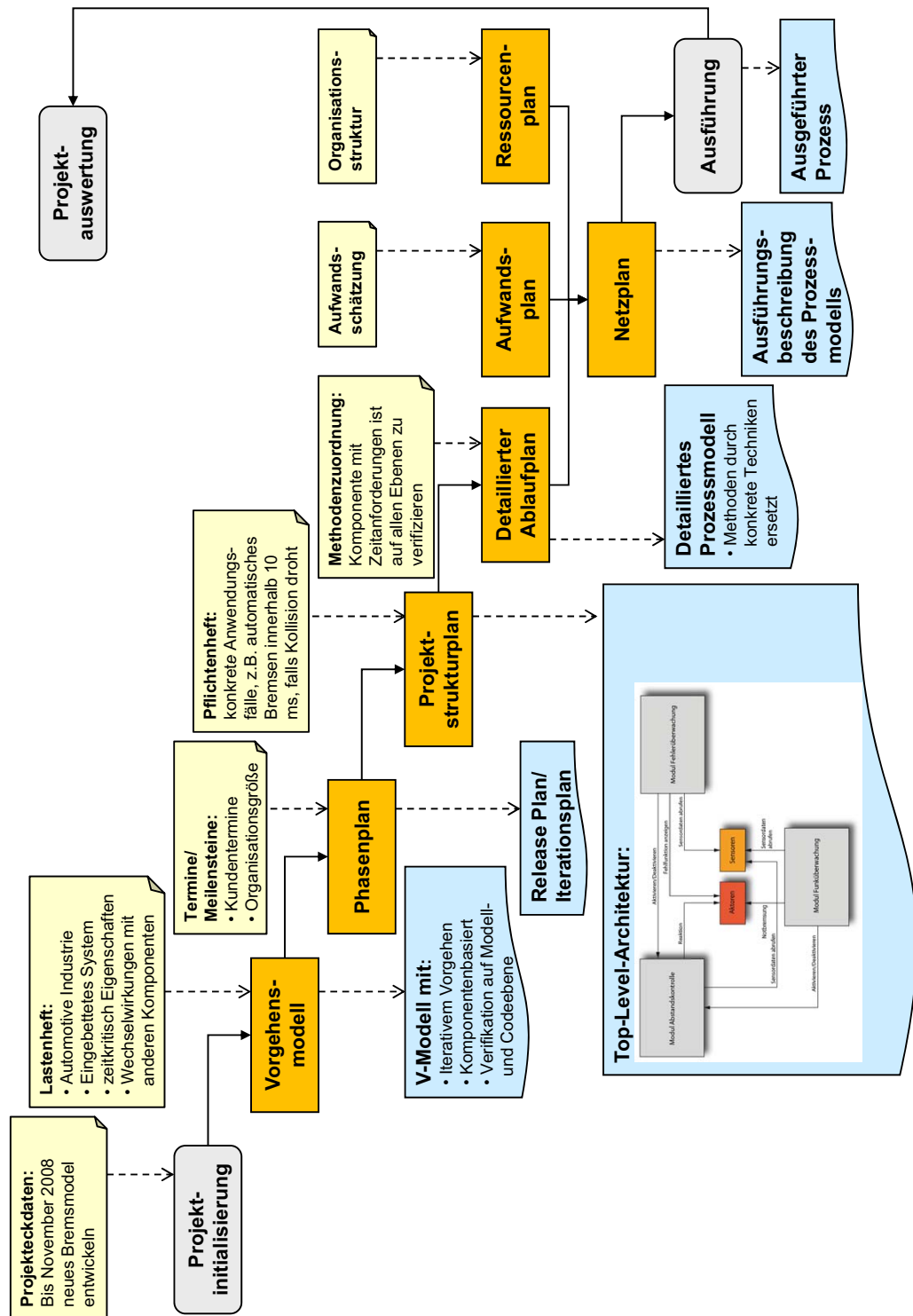


Abbildung 3.1: Aspekte des Lebenszyklus eines Softwareprojekts, die den Entwicklungsprozess beeinflussen

### 3.1 Die Herausforderungen laufzeitkritischer NFE an Vorgehensmodelle

allgemein üblich sind, dargestellt [Buh04]. Unterhalb der Phasen sind die jeweiligen Ergebnisdokumente und oberhalb die einfließenden Kriterien zu sehen. Der Schwerpunkt ist ebenfalls auf derartige Aktivitäten gelegt, die im engen Zusammenhang mit nicht-funktionalen Eigenschaften stehen.

Die erste Projektphase ist die Phase der Projektinitialisierung. In dieser werden die Projektziele bestimmt, die Machbarkeit geprüft, Risiken analysiert, Verträge ausgearbeitet sowie eine grobe Organisationsstruktur zur Projektrealisierung geschaffen. Zur Umsetzung dieser Aufgaben ist die Erstellung eines Lastenhefts (Software Requirements Specification) erforderlich. Das Lastenheft beschreibt die Anforderungen und Erwartungen des Nutzers an das System. Die vom Kunden erwarteten Qualitätseigenschaften sind im Beispiel die Reaktionen des Bremssystems bezüglich der Zeit. Diese sind als kritisch einzustufen und müssen daher als Risikofaktor sowohl in der Zeit- als auch der Kostenplanung beachtet werden. Dabei kommen an dieser Stelle, falls vorhanden, Meßwerte aus Metriken vorangegangener ähnlicher Projekte oder im einfachsten Fall Erfahrungswerte zum Einsatz.

Anhand der Projektedaten und den Anforderungen an das System wird ein geeignetes Vorgehensmodell ausgewählt und eventuell entsprechend angepasst (Tailoring). Im Beispiel wurde das V-Modell XT ausgewählt und an das iterative Vorgehen angepasst. Dies ist notwendig, da, wie später noch erläutert wird, die zeitkritischen Komponenten als Kern zuerst entwickelt werden müssen und das System dann stufenweise ausgebaut wird, um das Risiko der Überschreitung der Zeitanforderungen überwachen zu können. In dieser Phase kommen bereits die ersten Probleme bei der durchgängigen Unterstützung von Qualitätseigenschaften zum Vorschein. Da das Bremssystem mit anderen Komponenten des Autos kommuniziert, sind die Anforderungen mit diesen Komponenten abzustimmen. Deshalb sind Aspekte der komponentenbasierten Entwicklung zu beachten und entsprechende Methoden einzusetzen. Ansätze für NFE-Unterstützung in der komponentenbasierten Entwicklung befinden sich noch im Forschungsstadium und sind daher in standardisierten Vorgehensmodellen noch nicht integriert. Deshalb sind bereits erprobte Methoden aus anderen Quellen zu evaluieren. Die Auswahlentscheidung muss in den Tailoringprozess einfließen. Eine weitere problematische Tailoringentscheidung in dieser Phase ist die Integration von Verifikationstechniken für diese Eigenschaften. Im V-Modell XT ist dafür die Unterstützung nur sehr grob beschrieben, was daran liegt, dass die konkrete Umsetzung der Verifikation stark von den zu verifizierenden Eigenschaften abhängt. Für die weitere Planung ist es wichtig, an dieser Stelle zumindest einen Platzhalter für diese Aktivitäten zu haben. So kann deren Einfluss auf die Termin- und Meilensteinplanung und damit auf den Phasenplan berücksichtigt werden.

Die nächste Phase, in der NFE einen Einfluss auf die Planung haben, ist die Erstellung des Projektstrukturplans. Zu diesem Zeitpunkt wurde bereits das Pflichtenheft erstellt. Dieses wird aus den Anforderungen des Lastenhefts (auch grobes Pflichtenheft genannt) entwickelt. Laut DIN 69905 [DIN97] umfasst das Pflichtenheft die „vom Auftragnehmer erarbeiteten Realisierungsvorgaben aufgrund der Umsetzung des vom Auftraggeber vorgegebenen Lastenhefts“. Die Anforderungen sind nun präzisiert, vollständig und nachvollziehbar sowie mit technischen Festlegungen der Betriebs- und Wartungsumgebung verknüpft. Wie bereits in Kapitel 2 erläutert, sind nicht-funktionale Eigenschaf-

### 3 Die Idee des PROKRIS-Ansatzes

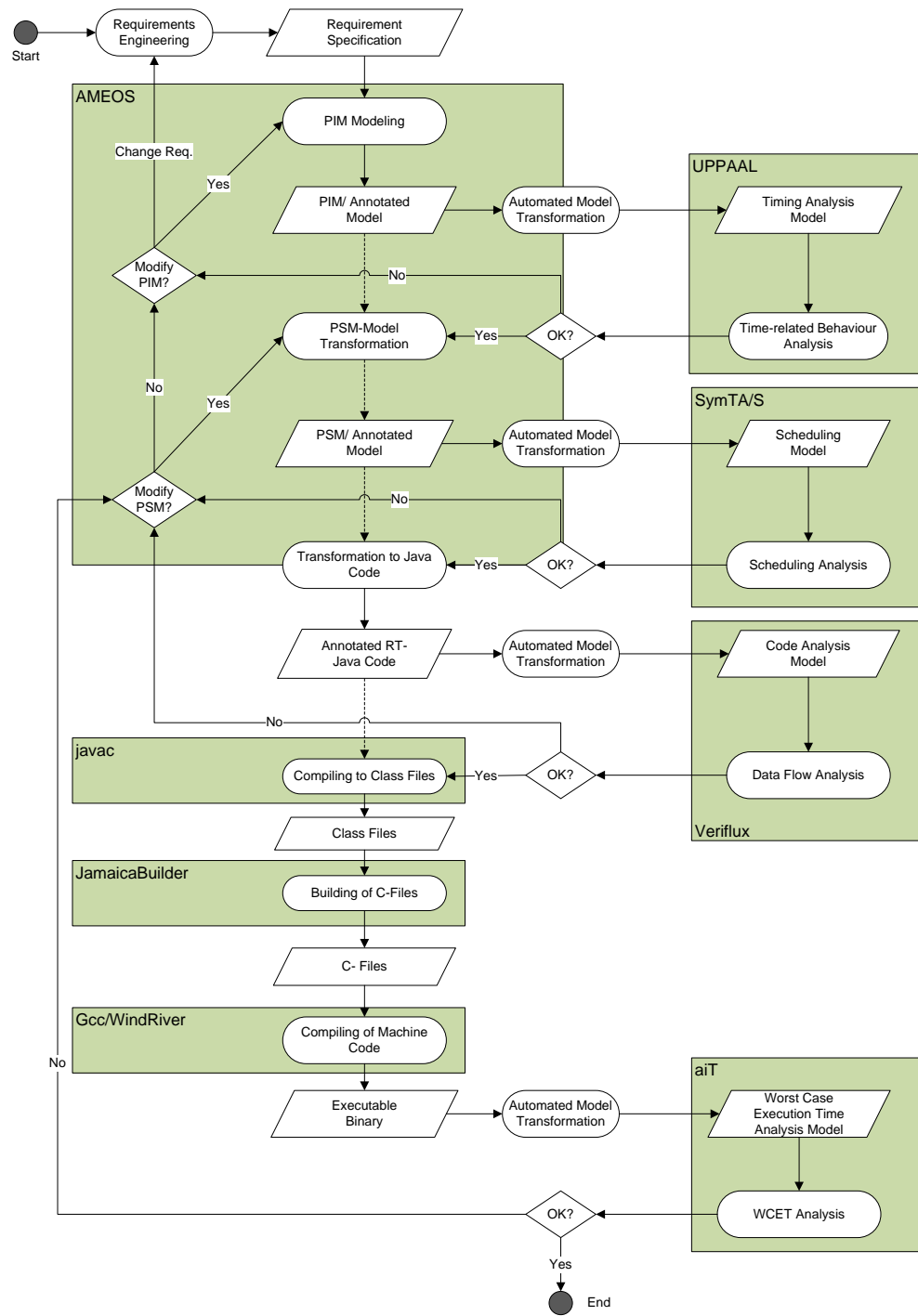


Abbildung 3.2: SuReal-Prozess für zeitkritische Komponenten



ten in ihrem Erfüllungsgrad optimierbar. Gerade deshalb sind explizite und testbare Grenzwerte der nicht-funktionalen Anforderungen festzulegen. Außerdem erfolgt deren Bindung an Funktionalität. So ist es möglich, dieses Wissen in die Erstellung der Top-Level-Architektur des Systems einfließen zu lassen. Im Beispiel kann das eigentliche Bremsmodul als die zeitkritische Komponenten des Projekts herauskristallisiert werden. Außerdem sind die Zeitanforderungen zu dekomponieren, d.h. auf die Subkomponenten des Bremssystems aufzuteilen.

Anhand der Komponentenstruktur kann nun ein detaillierter Ablaufplan erstellt werden. Dabei wird die Entwicklungsreihenfolge der Komponenten festgelegt. Außerdem ist nun genug Wissen über das System vorhanden, um die konkreten Techniken z.B. für die Modellierung und Verifikation zu bestimmen. Im Beispiel wird das SuReal-UML-Profil zur Modellierung der Zeitbedingungen eingesetzt. Zur Verifikation auf Modellebene wird die SuReal-spezifische Erweiterung des UPPAAL-Modelcheckers verwendet. Die Techniken der systemnahen Modellierung und der Implementierungsebenen werden ebenfalls festgelegt. Einen Überblick über das dadurch entstehende Prozessmodell für die zeitkritischen Komponenten gibt Abbildung 3.2.

Zusammen mit der Ressourcen- und Aufwandsplanung ist es nun möglich einen Netzplan zu erstellen, anhand dem das Entwicklungsprojekt durchgeführt wird. Außerdem sind die eingesetzten Techniken aufeinander abzustimmen, um eine Methoden- und Werkzeugumgebung für die Entwickler bereitzustellen.

Das Beispiel zeigt, dass sich die nicht-funktionalen Eigenschaften an mehreren Stellen auf die konkrete Ausprägung des Entwicklungsprozesses auswirken. Dabei verlangen sie innerhalb des Prozesses andere Lösungen als funktionale Anforderungen. Einige dieser Besonderheiten werden daher im nächsten Abschnitt diskutiert.

#### 3.1.2 NFE-spezifische Probleme des Lebenszyklus

##### Charakter nicht-funktionaler Eigenschaften

Wie bereits in Kapitel 2 bei der Definition nicht-funktionaler Anforderungen diskutiert wurde, unterscheiden sich diese in einigen grundlegenden Eigenschaften von funktionalen und sind daher im Entwicklungsprozess auch anders zu behandeln. Die grundlegenden Abweichungen sind die Optimierbarkeit der Anforderungen, deren eventuelle negative (oder auch positive) Korrelation untereinander, deren Operationalisierung sowie damit zusammenhängend deren Veränderung über den Entwicklungszeitraum.

Die Optimierbarkeit der Anforderungen ist vor allem für die Kosten- sowie die Risikoanalyse wichtig. In Bezug auf die Kostenanalyse lässt sich die Problematik anschaulich anhand des *Magischen Dreiecks* (Abbildung 3.3) darstellen. Jede Ecke beschreibt eine der Größen Qualität, Zeit und Kosten. Der Schnittpunkt des Dreiecks mit den Achsen gibt an zu welchem Grad der jeweilige Parameter erfüllt ist. Wird der Flächeninhalt konstant gehalten, bedingt die Verbesserung eines Parameters die Verschlechterung eines anderen. Eine Optimierung der Qualitätseigenschaften geht demnach immer mit einer Verschlechterung (also Erhöhung) von Zeit und/oder Kosten einher. Daher sind bei der Entwicklung Grenzen der nicht-funktionalen Anforderungen in Form von Ma-

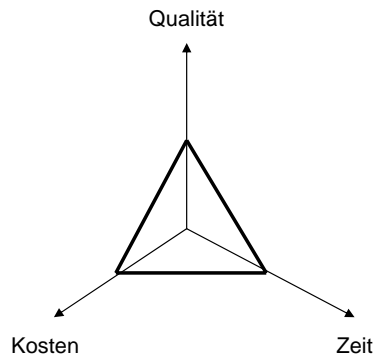


Abbildung 3.3: Magisches Dreieck des Projektmanagements nach [Hof08]

ßen festzulegen. Diese müssen explizit spezifiziert werden, um deren Kommunikation an die Entwickler zu ermöglichen. In die Kosten- und Risikoanalyse ist ebenfalls die Kritikalität der Eigenschaften (z.B. harte Echtzeitanforderungen bei lebenswichtigen Funktionen, wie im vorgestellten SuReal-Beispiel) mit einzubeziehen. Auch eventuelle negative Einflüsse nicht-funktionaler Eigenschaften untereinander sind zu analysieren. So wirken sich beispielsweise Geschwindigkeitsoptimierungen eines Systems auf andere Qualitätseigenschaften wie Zuverlässigkeit, Wartbarkeit und Transparenz meist negativ aus und sind daher auf zeitkritische Systemkomponenten zu beschränken [Hof08].

Nicht-funktionale Eigenschaften sind als Beschränkungen von funktionalen Eigenschaften definiert. Das bedeutet, dass diese während des gesamten Prozesslebenszyklus auch explizit in Bezug zu einer Systemfunktion spezifiziert werden müssen. Eine Problematik, die sich dabei aufzeigt, ist die Veränderung der nicht-funktionalen Eigenschaften über den Entwicklungszyklus. Wie bereits diskutiert, müssen nicht-funktionale Eigenschaften auf einer gewissen Abstraktionsstufe des Systems operationalisiert sein. Dass ist bei Sicherheitsanforderungen offensichtlich, z.B. erhält man einen einfachen Datenschutz schon über das Integrieren einer Passwortabfrage, die rein funktional ist. Doch auch harte Zeitanforderungen, wie beim Bremssystem der Beispielanwendung, können letztendlich nur durch Reservieren von CPU-Zeit und Speicherplatz zur Laufzeit garantiert werden [GPA<sup>+</sup>04]. Während der Entwicklung müssen also dementsprechende Abbildungsmechanismen untersucht werden. Dazu sind Techniken in das Vorgehen zu integrieren, die diese Abbildungen im Vorfeld untersuchen. Eine Möglichkeit ist z.B. die Nutzung der Techniken des NFR-Frameworks [TC99]. Das Framework erlaubt es, nicht-funktionale Eigenschaften explizit zu repräsentieren sowie verschiedene Designentscheidungen anhand von Softgoal-Graphen zu durchdenken und zu bewerten. Das dabei entstehende Wissen ist zu einem gewissen Teil wiederverwendbar. Daher gibt es erste Ansätze, dieses in Form von Mustern (z.B. [FCH06]) bzw. Ontologien (z.B. [SJP<sup>+</sup>07]) bereitzustellen.

Zusammenfassend ist zu sagen, dass der Charakter nicht-funktionaler Eigenschaften einen großen Einfluss auf den Entwicklungsprozess hat. So sind die Anforderungen ex-

plizit zu erfassen und deren Entwicklung zu dokumentieren, um die Verfolgbarkeit und Kommunikation zu gewährleisten. Außerdem müssen zusätzliche spezifische Techniken eingesetzt werden, um die durchgängige Umsetzung der Eigenschaften zu gewährleisten. Dabei ist deren unterschiedliche Ausprägung auf den verschiedenen Abstraktionsebenen zu beachten.

#### **Makro- und Mikroprozesse**

Das Beispielprojekt verdeutlicht, dass die Spezifikation des Prozesses mehrere Stufen durchläuft. So werden bereits bei der Definition des Vorgehens Maßnahmen zur Absicherung der Zeitbedingungen integriert. In diesem Fall ist das die Integration von Verifikationstechniken sowohl auf Modell- als auch auf Codeebene. Dabei werden noch keine konkreten Techniken, wie z.B. die Nutzung des Modelcheckers UPPAAL zum Verifizieren von UML-Modellen, welche um Zeitspezifikationen mittels des SuReal-UML-Profiles erweitert sind, definiert. Ziel ist vorerst nur die Aufwandsschätzung der benötigten Ressourcen (Personal, Werkzeuge, Zeit). Das Modell des Prozesses befindet sich dabei auf der Makroebene (Kapitel 2). Die konkreten Techniken der Verifikation, wie beschrieben, werden erst nach genauerer Analyse der Anforderungen beim Aufstellen der Mikroprozesse, also der detaillierten Ablaufplanung, festgelegt. Diese Festlegung auf eine bestimmte Technik ist auch erst zu diesem Zeitpunkt sinnvoll, da die eingesetzten Techniken sowohl von der Art der nicht-funktionalen Eigenschaften (Realtime vs. Sicherheit), als auch von anderen Bedingungen, wie dem Aufwand bezüglich zu den Kosten sowie der Organisationsstruktur und dem Erfahrungspotential des Teams abhängen.

Als Beispiel wurde bisher die Integration von Verifikationstechniken angeführt. Die durchgängige Unterstützung von NFE verlangt aber weitere Methoden innerhalb des Prozesses, wie die explizite Modellierung der nicht-funktionalen Anforderungen oder deren Dekomposition. So ist eine vage Spezifikation, wie „Das Bremssystem muss schnell sein.“ unzureichend. Für die systematische Umsetzung derartiger Anforderungen ist die Angabe expliziter Werte für die Aussage „schnell“ notwendig. Außerdem muss die Anforderung auf konkrete Funktionen des Bremssystems heruntergebrochen (dekomponiert) werden. Auch für diese geforderten Prozessschritte und Methoden gibt es unterschiedliche Ausprägungen (Verfahren bzw. Techniken <sup>1</sup>) für verschiedenartige Eigenschaften. Für die Modellierung gibt es beispielsweise eine Vielzahl von UML-Profilen.

Es existieren demnach zwei Arten von Prozessen, deren Unterschied in der Bereitstellung von allgemeinen Methoden bzw. konkreten Techniken liegt. Daher kann man von Makro- und Mikroprozessen sprechen (Kapitel 2). Im Makroprozess werden zur NFE-Umsetzung notwendige Methoden innerhalb eines Vorgehensmodells definiert. Erst auf Mikroprozessebene, während der detaillierten Projektplanung, werden dafür konkrete Techniken festgelegt. Wie bereits erwähnt, gibt es derzeit Ansätze auf beiden Abstraktionsebenen. Ziel sollte es sein, diese Modelle über die Abstraktionsebenen hinweg wiederzuverwenden.

Im Kontext der Integration von Maßnahmen zur Umsetzung nicht-funktionaler Eigenschaften in Vorgehensmodellen ergibt sich die in Abbildung 3.4 dargestellte Begriffswelt. Die in den Abbildungen eingeführte farbige Unterlegung der verschiedenen Abstraktions-

---

<sup>1</sup>nach Definition in Kapitel 2.3

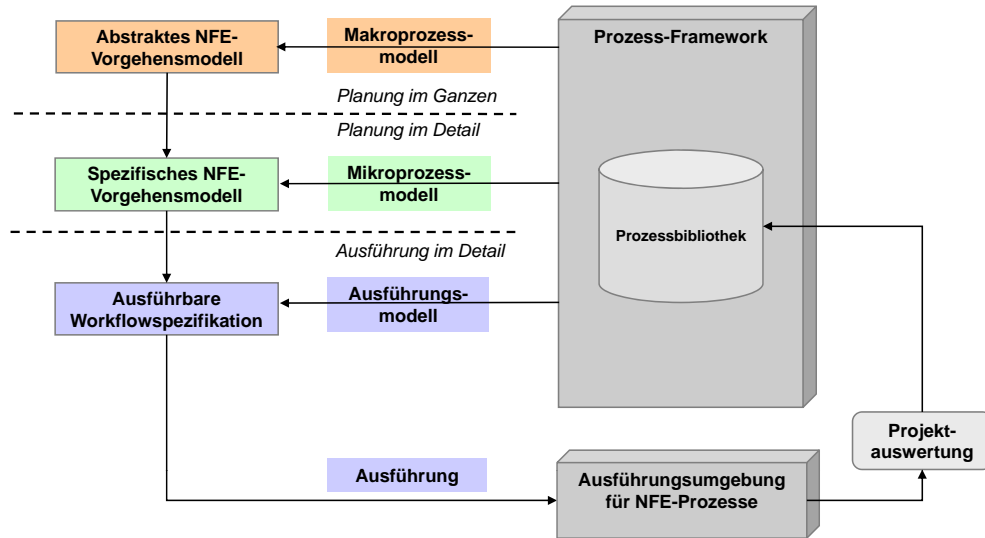


Abbildung 3.4: Abstraktionsebenen von Vorgehensmodellen

stufen wird innerhalb der Arbeit weiterverwendet, um die Orientierung zu erleichtern.

Es werden drei Abstraktionsstufen einer Prozessspezifikation unterschieden: Makroprozessmodell, Mikroprozessmodell und Ausführungsmodelle. Bei den Makroprozessen handelt es sich um *Abstrakte NFE-Vorgehensmodelle*, da sie das Vorhandensein nicht-funktionaler Anforderungen auf einer generischen Ebene beachten. Entsprechend sind Mikroprozesse als *Spezifische NFE-Vorgehensmodelle*<sup>2</sup> anzusehen. Zur Ausführung des Mikroprozesses in einem konkreten Projekt muss dieser in einem weiteren Schritt an organisationsspezifische Bedingungen, wie verfügbare Mitarbeiter, vorhandene Werkzeuge und Dokumentenverwaltungen angepasst werden. Damit erhält man das *Ausführungsmodell*. Diese verschiedenen Vorgehensmodelle sind demnach verschiedene Detaillierungsgrade von ein und demselben Vorgehensmodell und besitzen daher eine Verfeinerungsbeziehung untereinander. Um diese zu unterstützen, sollten die Modelle in ein Prozess-Framework integriert sein, deren zentrale Basis eine Prozessbibliothek darstellt.

#### Ausführungsunterstützung

Bei der Entwicklung kritischer Systeme kommt es gezielt darauf an, die nicht-funktionalen Anforderungen an das System zu erfüllen. Dabei handelt es sich nicht um ein additives Zusammensetzen von Funktionalität, sondern um optimierbare Eigenschaften des Systems. Es ist eine Kostenfrage, wie weit diese Optimierung getrieben werden soll. Außerdem ist es unabdingbar, diese Eigenschaften in den Fokus der Entwicklung zu legen. Um Fehlentwicklungen zu vermeiden, ist dabei eine gewisse Führung der Entwickler

<sup>2</sup>Genau genommen handelt es sich um das NFE-spezifische NFE-Vorgehensmodell. Vereinfacht werden in dieser Arbeit dafür synonym die Begriffe spezifisches NFE-Vorgehensmodell und NFE-spezifisches Vorgehensmodell eingesetzt.

durch einen Prozess unerlässlich. So muss sichergestellt werden, dass Komponenten mit den kritischen Eigenschaften höhere Priorität bei der Entwicklung erlangen, dass die Eigenschaften kontinuierlich verifiziert bzw. getestet werden sowie, dass die Eigenschaften und deren Umsetzungen verfolgt und kommuniziert werden. Letzteres ist vor allem in größeren Entwicklerteams gezielt zu steuern. So haben Studien gezeigt, dass ein wesentliches Problem bei der durchgängigen Behandlung nicht-funktionaler Eigenschaften auf deren unzureichenden Kommunikation über Hierarchie- und Organisationsebenen hinweg besteht (z.B. [Bor04], [ArE96]). Sind dazu noch verteilte Teams an der Entwicklung beteiligt, sollte es das Ziel sein, zur Steuerung der Entwicklung eine *automatisierte Prozessausführung* zu etablieren. In Abbildung 3.4 ist diese Ausführungsumgebung dargestellt.

Dieser Ansatz wird von mehreren Autoren untermauert. Neben den bereits in Kapitel 2 beschriebenen Techniken zur Prozessbewertung, wie CMMI, welche definierte Prozessmodelle fordern, diskutiert Sommerville in [Som07] die Notwendigkeit verlässlicher Prozesse für die Entwicklung kritischer Systeme. Verlässliche Prozesse sind wohl definiert und nachvollziehbar. Wohl definiert ist ein Prozess, der standardisiert und dokumentiert ist. Nachvollziehbare Prozesse müssen unabhängig von individueller Interpretation der am Prozess beteiligten Personen sein. Dies kann durch eine maschinelle Unterstützung des Prozesses erreicht werden. Wallmüller führt diese Unterstützung sogar als Qualitätsmerkmal für die Beurteilung konkreter Vorgehen ein [Wal01]. Daraus ergeben sich mehrere Vorteile. Im SuReal-Projekt könnte so die Verifikation auf den einzelnen Modellebenen obligatorisch integriert werden. Unterstützbar wäre ebenso die Kommunikation der Zeitanforderungen an die Gesamtkomponente des Bremssystems gegenüber den Entwicklern der Subsysteme. So kann ihnen ihre Verantwortung innerhalb der Entwicklung bewusster gemacht werden. Des weiteren werden bei der Entwicklung kritischer Systeme verschiedenartige Werkzeuge zur Modellierung, Verifikation, Simulation oder Generierung eingesetzt. Mit Hilfe einer prozessbasierten Ausführungsunterstützung wird es möglich, eine Werkzeugkette aufzubauen und somit einen sinnvollen Einsatz der Werkzeuge sicherzustellen.

Aber nicht nur in Bezug auf die Prozesseigenschaften an sich ist eine automatisierte Prozessunterstützung sinnvoll. Techniken zur Umsetzung von nicht-funktionalen Anforderungen erfordern unter Umständen ein hohes Erfahrungspotential. Außerdem ist der Erfolg eines Vorgehensmodells abhängig vom Verständnis des Entwicklers. Er muss sowohl durch Schulungen als auch durch andere Hilfsmittel in der Lage sein, sowohl das notwendige Wissen über den gesamten Entwicklungsprozess als auch über Methoden, Techniken und Werkzeuge zu erlangen [Wal01]. Ist eine automatisierte Prozesssteuerung vorhanden, können einzelne Prozessschritte gezielt mit Methodenwissen hinterlegt werden. Das können z.B. Anleitungen, Beispiele oder ein Überblick über die Einordnung der Aufgabe in das Gesamtverfahrenmodell sein.

#### **Agilität vs. Formalisierung von Vorgehensmodellen**

Eine gegenläufige Tendenz ist die Entwicklung im Bereich der Vorgehensmodelle weg von starren unflexiblen Modellen hin zu agilen Modellen. Prinzipiell ist dieser Ansatz sowohl bezüglich der Reaktionsfähigkeit auf geänderte Kundenwünsche als auch auf ein

selbstbestimmtes und damit erfülltes Arbeiten der Entwickler als positiv zu bewerten. Diese Entwicklung hat im Bereich sicherheits- oder zeitkritischer Systeme ihre Grenzen. Bereits im vorhergehenden Abschnitt wurde auf die Notwendigkeit verlässlicher Prozesse für die Entwicklung kritischer System hingewiesen. Dies ist insbesondere durch eine starke Formalisierung des Vorgehens zu erreichen, welche in agilen Vorgehen nicht vorgesehen ist. Ebenfalls an ihre Grenzen stoßen die agilen Methoden bei der Koordination größerer Teams, die eventuell sogar noch verteilt arbeiten. Andererseits ist es im Kontext nicht-funktionaler Eigenschaften nur schwer, die Definition eines allgemeingültigen formalisierten Modells zu erreichen. Wie bereits mehrfach beschrieben, besitzen Prozesse in der betrachteten Domäne einen hohen Individualisierungsgrad. Dies ist bedingt durch den stark heterogenen Charakter nicht-funktionaler Eigenschaften. Dadurch ist es erst nach einer genauen Analyse der nicht-funktionalen Anforderungen möglich Prozessaktivitäten mit konkreten Techniken zu hinterlegen. Ein weiterer Gesichtspunkt sind die am Prozess beteiligten Personen, welche als Experten auf ihrem Gebiet anzusehen sind. Für deren Motivation ist eine zu starke Formalisierung eher kontraproduktiv. Hier gilt es also Vor- und Nachteile gezielt abzuwägen.

## 3.2 Anforderungen an die systematische Prozessunterstützung der Entwicklung laufzeitkritischer Systeme

Zusammenfassend lassen sich folgende Anforderungen an die systematische Prozessunterstützung für die Entwicklung laufzeitkritischer Softwaresysteme festlegen:

1. Spezifikation notwendiger Erweiterungen von Vorgehensmodellen zur systematischen Unterstützung laufzeitkritischer Eigenschaften
2. Basierend auf den Konzepten der Wiederverwendung und der Differenzierung verschiedener Abstraktionsstufen von Vorgehensmodellen ist eine entsprechende Methodik zur Modellierung von Vorgehensmodellen für die Entwicklung laufzeitkritischer Systeme mit folgenden Eigenschaften notwendig:
  - Unterstützung der Wiederverwendung von Vorgehensmodellen und deren Fragmenten
  - Verbindung der Abstraktionsstufen entsprechend der Konkretisierung der nicht-funktionalen Eigenschaften innerhalb der Systementwicklung, um die adaptive Erstellung eines Entwicklungsprozesses zu ermöglichen
  - Unterstützung der Modellierung von ausführbaren Werkzeugketten
3. Die Entwicklung der technischen Grundlagen und einer Werkzeugunterstützung für die Bereitstellung, Verfeinerung und Ausführung der erweiterten Vorgehensmodelle zur Entwicklung kritischer Systeme.

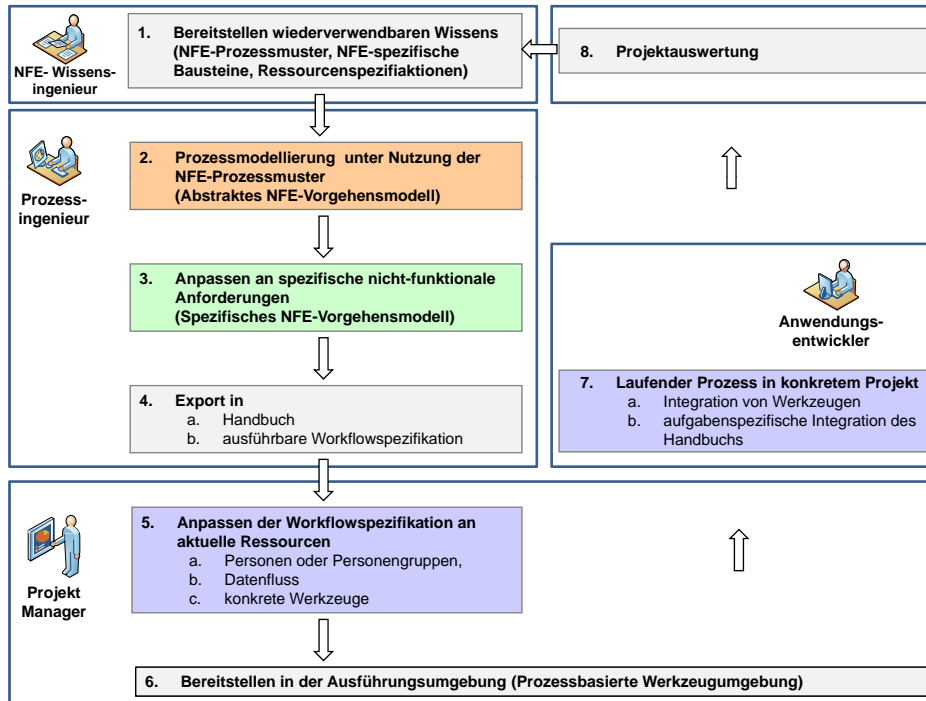


Abbildung 3.5: Rollenbasiertes Einsatzszenario der PROKRIS-Methodik

### 3.3 Der PROKRIS-Ansatz als Lösung

Diese Arbeit stellt das *PROKRIS-Framework* sowie die Methodik der *kontextbasierten Anpassung von Vorgehensmodellen für laufzeitkritische Softwaresysteme* als Lösung vor. Die Methodik wird kurz als *PROKRIS-Methodik* bezeichnet. Bei der Anwendung der Methodik handelt es sich um ein neues mentales Modell der Prozessmodellierung. Der folgende Abschnitt gibt einen Überblick über die grundlegenden Idee des PROKRIS-Ansatzes sowie die Architektur des PROKRIS-Frameworks.

#### Rollenbasierter Einsatz der PROKRIS-Methodik

Abbildung 3.5 stellt schematisch die Nutzung der PROKRIS-Methodik innerhalb eines Projekts zur Entwicklung eines laufzeitkritischen Softwaresystems durch verschiedene Rollen dar. Das PROKRIS-Framework erweitert die Architektur eines allgemeinen Prozess-Frameworks (Kapitel 2.7). Daher sind die Rollen entsprechend benannt.

Aufgabe des NFE-Wissensingenieurs (Abbildung 3.5) ist die Bereitstellung von wiederverwendbarem Prozesswissen (Punkt 1.). In der PROKRIS-Methodik ist dieses Wissen auf der Ebene der abstrakten NFE-Vorgehensmodelle in vordefinierten NFE-Prozessmustern beschrieben. Diese Muster beschreiben in Form von Methoden, die bezüglich der NFE generisch sind, Vorgehensmodellerweiterungen, die für die systematische Umsetzung nicht-funktionaler laufzeitbasierter Anforderungen notwendig sind. Auf der NFE-spezifischen Ebene können diese NFE-Prozessmuster an die Spezifika konkreter

### 3 Die Idee des PROKRIS-Ansatzes

NFE angepasst werden, indem die abstrakten Methodenbeschreibungen durch konkrete Techniken und Verfahren instanziiert werden. Diese werden in NFE-spezifischen Prozessbausteinen modelliert. Außerdem wird eine organisationsspezifische Ressourcenspezifikation zur Anpassung der ausführbaren Workflowspezifikation benötigt.

Der Prozessingenieur ist für das Zusammenstellen (Punkt 2.) und die gezielte Anpassung der erweiterten Vorgehensmodelle an verschiedene nicht-funktionale Laufzeiteigenschaften (Punkt 3.) verantwortlich. Er ist so in der Lage Vorgehensbeschreibungen sowohl auf der Makro- als auch auf der Mikroebene bereitzustellen, wobei er den Makroprozess als Ausgangspunkt der Anpassung nimmt.

In einem weiteren Schritt kann das Vorgehensmodell teilweise oder vollständig in eine ausführbare Spezifikation übersetzt werden und durch den Projektmanager an organisationsspezifische Bedingungen, wie verfügbare Mitarbeiter, vorhandene Werkzeuge und Dokumentenverwaltungen angepasst werden (Punkt 5.). Die dazu notwendigen Informationen hat der NFE-Wissensingenieur in der Ressourcenspezifikation bereitgestellt. Für die Ausführung steht eine prozessbasierte Ausführungsumgebung zur Verfügung, an der sich die Entwickler anmelden können und gezielte Aufgaben innerhalb des Vorgehens zugeordnet bekommen. Diese Ausführungsumgebung überwacht und steuert die Entwicklung, kann Werkzeugaufrufe initiieren und ermöglicht die Weitergabe von Daten anhand des vorab modellierten Datenflusses. Sie bietet außerdem gezielte Hilfestellungen zu den Aufgaben an. So steht den Entwicklern eine individuell angepasste prozessbasierte Werkzeugumgebungen zur Verfügung, mit denen eine systematische Entwicklung laufzeitkritischer Systemteile möglich ist (Abbildung 3.5, Punkt 7.).

Die grundlegende Idee bei der Entwicklung des PROKRIS-Ansatzes ist, bestimmte Freiheitsgrade in der Prozessmodellierung so lange wie möglich offen zu lassen und diese erst zum spätest möglichen Zeitpunkt zu entscheiden, wie es in [PP03] gefordert wird. Dies wird in der Trennung der Abstraktionsebenen und den jeweiligen Zielen der einzelnen Ebenen deutlich. Innerhalb der Prozessausführung ist es dem Entwickler möglich, aus vorab definierten Gruppen möglicher Ressourcen, z.B. verschiedener UML-Werkzeuge, erst zur Laufzeit sein präferiertes Werkzeug zu wählen. Zudem ist es den Entwicklern an vordefinierten Entscheidungspunkten möglich, den weiteren Ablauf interaktiv zu beeinflussen.

Über die Projektauswertung (Abbildung 3.5, Punkt 8.) kann der Wissensingenieur die Beschreibung des wiederverwendbaren Prozesswissens kontinuierlich verbessern. Verantwortlich für das notwendige Feedback ist nicht nur der Projektmanager, sondern auch die Entwickler und der Prozessingenieur.

#### **Architektur des PROKRIS-Frameworks**

Die PROKRIS-Methodik für die Erstellung und Nutzung der erweiterten Vorgehensmodelle für laufzeitkritische Anwendungen wird durch das PROKRIS-Framework unterstützt. Abbildung 3.6 zeigt schematisch die Komponenten des Frameworks und ihre Zusammenarbeit.

Den Kern des Frameworks bildet eine Prozessbibliothek. Die *PROKRIS-Prozessbibliothek* ist auf die Verwaltung der NFE-spezifischen Prozessbausteine und NFE-Prozessmuster der NFE-Bausteinlandschaft sowie der Ressourcenspezifikationen der



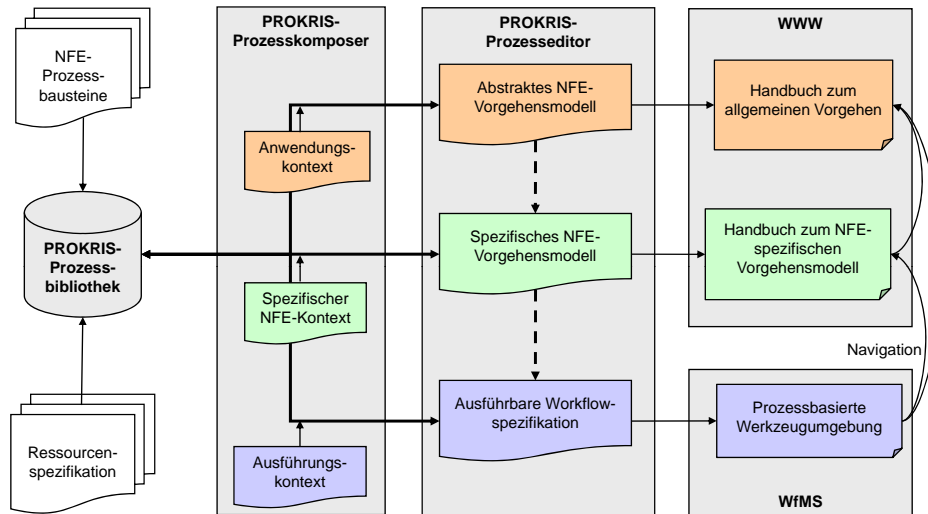


Abbildung 3.6: Hauptbestandteile des PROKRIS-Frameworks

Organisation spezialisiert.

Die Strukturierung des Inhalts der Prozessbibliothek erfolgt über eine Facettenklassifikation, welche durch den Kompositionsfilter des *PROKRIS-Prozesskomposers*<sup>3</sup> für die Erstellung und Anpassung der verschiedenen Vorgehensmodelle genutzt wird. Der Kompositionsfilter filtert anhand verschiedener Projektkontexte die Prozessbausteine sowie deren Beziehungen untereinander und stellt entsprechende Sichten auf die Prozessbibliothek bereit. Mithilfe dieser kann der Prozessingenieur NFE-Vorgehensmodelle komponieren und verfeinern und im *Prozesseditor* durch zusätzliche Informationen erweitern bzw. anpassen. Die verschiedene Farbigkeit der Eingangsdaten in Abbildung 3.6 im PROKRIS-Prozesskomposer symbolisiert den Einfluss der verschiedenen Kontexte auf die Zusammenstellung der einzelnen Abstraktionsebenen des erweiterten Vorgehensmodells.

Der PROKRIS-Prozesseditor kann auf der Funktionalität üblicher Autorenwerkzeuge für Prozess-Frameworks, wie dem Eclipse Prozess-Framework Composer (EPF Composer, [EPF]) aufbauen. Folgende Dialogmechanismen müssen zusätzlich zur Verfügung gestellt werden:

- Bearbeitung der Ressourcenspezifikationen
- Editieren der Facettenklassifikation
- Zuordnung der NFE-Prozessbausteine zur Facettenklassifikation
- Nutzung des Kompositionsfilters während der Modellierung und Anpassung des erweiterten Vorgehensmodells

<sup>3</sup>Der Begriff „Komposer“ ist vom englischen Begriff „Process Composer“ abgeleitet, für den bisher keine Übersetzung bekannt ist.

Interessant ist die Nutzung der NFE-spezifischen Vorgehensmodelle während der laufenden Entwicklung (zusätzlich zu den üblichen Einsatzzwecken von Prozessen). Abbildung 3.6 (rechts) zeigt die Unterstützung des Exports in ein webbasiertes Nutzerhandbuch. Zum Einsatz der mit dem PROKRIS-Komposer erstellten NFE-spezifischen Vorgehensmodelle innerhalb eines konkreten Softwareentwicklungsprojekts steht ein weiteres Werkzeug, die *PROKRIS-Ausführungsumgebung*, zur Verfügung. Diese basiert auf der Architektur eines Workflow Management Systems (WfMS) und ermöglicht das Bereitstellen einer integrierten prozessbasierten Entwicklungsumgebung, welche die systematische Entwicklung mittels einer Werkzeugkette unterstützt. Der Entwickler sieht so seine Aufgaben, kann aus der Umgebung heraus die entsprechenden Werkzeuge starten und erhält spezifische Hilfen durch die Einbindung des Nutzerhandbuchs in die Umgebung. Durch Verknüpfung der Aufgaben innerhalb des ausgeführten Prozesses mit den entsprechenden Einträgen im Online-Handbuch können die Entwickler auf zugeschnittene Hilfen zugreifen. Eine Unterstützung des kollaborativen Arbeitens in verteilten Entwicklerteams ist ebenso denkbar. Weiterhin wird eine bessere Verfolgbarkeit von Anforderungen durch die Überwachung globaler Qualitätsanforderungen bei der parallelen Entwicklung von zusammenspielenden Komponenten unterstützt.

## 3.4 Einsatzszenarien und Rollen

Die Nutzung des PROKRIS-Frameworks erfolgt durch verschiedene Rollen, welche in die Bereitstellung oder Nutzung der erweiterten NFE-Vorgehensmodelle involviert sind. Im Folgenden werden die typischen Nutzungsszenarien rollenbezogen zusammengefasst. Prinzipiell können drei verschiedene Nutzungsszenarien unterschieden werden: das Bereitstellen von Prozesswissen, die Modellierung der erweiterten Vorgehensmodelle sowie die Ausführung eines Prozesses. Dabei lassen sich vier Hauptrollen unterscheiden, wie sie in Abbildung 3.5 dargestellt sind: Wissensingenieur, Prozessingenieur, Anwendungsentwickler und Projektmanager.

### 3.4.1 Bereitstellung von wiederverwendbarem Prozesswissen

#### Wissensingenieur (Knowledge engineer)

Die Aufgabe des Wissensingenieurs ist die Bereitstellung und Wartung des Inhalts der Prozessbibliothek. Er stellt neue Prozessbausteine zur Verfügung, kann Anpassungen an bestehenden Bausteinen vornehmen und Inhalte entfernen. Zusätzlich hat er die Möglichkeit innerhalb des Frameworks zu navigieren, sowie nach Prozessbausteinen zu suchen, diese zu kopieren, neu zuzuordnen und zu erweitern. Bei dieser Aufgabe unterstützt ihn der PROKRIS-Prozesseditor, welcher die notwendige Funktionalität bereitstellt. Sein Wissen zieht er sowohl aus dem Feedback der anderen Stakeholder als auch aus neu entwickelten Konzepten der Forschung und Entwicklung. Diese Rolle existiert identisch in allgemeinen Prozess-Frameworks.

#### **NFE-Wissensingenieur (NFE-Domain Engineer)**

Das PROKRIS-Framework benötigt eine zusätzliche Rolle zur Entwicklung von NFE-spezifischen Techniken, den NFE-Wissensingenieur. Da er von der Rolle „Wissensingenieur“ abgeleitet ist, besitzt er innerhalb des Frameworks alle Möglichkeiten wie dieser. Die grundlegende Aufgabe des NFE-Wissensingenieurs ist es, Implementierungen der NFE-Prozessmuster in Form von NFE-Prozessbausteinen bereitzustellen. Innerhalb des PROKRIS-Prozessframeworks muss er diese in die PROKRIS-Facettenklassifikation einordnen, um sie dem Filteralgorithmus zur Komposition zugänglich zu machen.

### **3.4.2 Modellierung abstrakter und NFE-spezifischer Vorgehensmodelle**

#### **Prozessingenieur (Process Engineer)**

Der Prozessingenieur nutzt das Framework, um Entwicklungsprozesse zu konstruieren. Dazu hat er über den PROKRIS-Prozesseditor die Möglichkeit innerhalb des Frameworks zu navigieren sowie nach Prozessbausteinen zu suchen, diese zu kopieren, neu zuzuordnen und zu erweitern. Dies ermöglicht ihm das Komponieren eines Vorgehensmodells. Der Prozessingenieur setzt das PROKRIS-Framework auf den beiden oberen Modellierungsstufen der NFE-Vorgehensmodelle ein. Dazu ist es ihm möglich, mit dem Kompositionsfilter spezifische Bausteine innerhalb der einzelnen Ebenen zu filtern.

#### **Projektmanager (Project Manager)**

Die Bereitstellung und Anpassung der ausführbaren Workflowspezifikation ist dagegen Aufgabe des Projektmanagers. Er kann bereitgestellte NFE-spezifische Vorgehensmodelle als Dokumentation (z.B. als Onlineversion mit entsprechender Navigationsstruktur), Vorlage für ein Planungssystem (z.B. Microsoft Project) oder ausführbare Beschreibung zur Nutzung innerhalb der prozessbasierten PROKRIS-Ausführungsumgebung exportieren. Außerdem passt er bereitgestellte Prozesse für ein Projekt an. Dazu verknüpft er den Prozess mit organisatorischen Informationen, wie Mitarbeitern, konkreten Werkzeugen und Terminen. Diese wählt er aus dem Ressourcenmodell der Organisation entsprechend des Ausführungskontexts aus.

### **3.4.3 Prozessanwendung innerhalb konkreter Projekte**

#### **Projektmanager (Project Manager)**

Die erstellten Vorgehensmodelle können durch den Projektmanager innerhalb eines Projekts in zwei Varianten eingesetzt werden. Einerseits kann er das Vorgehen als Handbuch (z.B. in Form einer Website oder eines Wiki) bereitstellen. Andererseits hat er die Möglichkeit, projektkritische Teile über die prozessbasierte PROKRIS-Ausführungsumgebung gesteuert ablaufen zu lassen. Dazu kann er Teile des Prozessmodells innerhalb dieser Umgebung bereitstellen und starten. Außerdem kann der Projektmanager über eine Monitoring-Schnittstelle den Ablauf überwachen.

#### Anwendungsentwickler (Application Engineer)

Der Projektmitarbeiter ist Nutzer des Prozesshandbuchs und der PROKRIS-Ausführungsumgebung. Die Ausführungsumgebung ordnet ihm konkrete Aufgaben innerhalb des Projektes zu. Darüber hinaus stellt diese eine prozessgesteuerte integrierte Werkzeugumgebung zur Verfügung. Das bedeutet, dass der Anwendungsentwickler die benötigten Werkzeuge direkt aus der Ausführungsumgebung heraus starten kann. Die Umgebung stellt ihm die notwendigen Daten zur Verfügung und ist in der Lage vorkonfigurierte Werkzeugketten automatisch auszuführen. Durch die Kopplung von zugeordneten Aufgaben mit dem Onlinehandbuch kann er dieses nutzen, um gezielt Informationen und Hilfestellungen zu Aufgaben, Arbeitsprodukten oder Werkzeugen zu erhalten.

### 3.5 Zusammenfassung

Das Kapitel stellte einen typischen Prozesslebenszyklus für ein Projekt mit zeitkritischen Eigenschaften vor. Anhand diesem wurden die spezifischen Herausforderungen an derartige Prozesse diskutiert. Dazu gehören der Charakter nicht-funktionaler Anforderungen, der sich in einigen wesentlichen Eigenschaften stark von funktionalen Anforderungen unterscheidet. Desweiteren wurden die verschiedenen Ausprägungsarten der Prozesse (Makro-, Mikro- und ausführbare Prozesse) diskutiert. Außerdem wurden die Vor- und Nachteile einer Prozessunterstützung innerhalb einer Ausführungsumgebung im Zusammenhang der Formalisierung bzw. Agilität von Prozessbeschreibungen erörtert. Als Schlussfolgerung dieser Diskussion wurden die Anforderungen an eine systematische Prozessunterstützung für die Entwicklung laufzeitkritischer Softwaresysteme spezifiziert.

Zur Lösung der dargestellten Probleme wurde das PROKRIS-Framework vorgestellt. Das Framework führt die *Kontextbasierte Anpassung von Vorgehensmodellen für laufzeitkritische Software* (kurz: *PROKRIS-Methodik*) als neue Methodik sowie eine unterstützende Werkzeugarchitektur ein. Zur Unterstützung der PROKRIS-Methodik ist es notwendig, die Architektur von allgemeinen Prozess-Frameworks zu erweitern. Um die Modellierung der erweiterten Vorgehensmodelle zur Entwicklung laufzeitkritischer Systeme zu ermöglichen, muss sowohl die Funktionalität der Prozessbibliothek ausgebaut, als auch eine neue Komponente, der PROKRIS-Prozesskomposer, eingeführt werden. Zur Ausführung der erweiterten Vorgehensmodelle in einem konkreten Projekt dient die PROKRIS-Ausführungsumgebung. Diese ermöglicht die Bereitstellung einer prozessunterstützten Werkzeugumgebung für die Entwickler.

Das Kapitel gab außerdem einen Überblick darüber, wie verschiedene Rollen das PROKRIS-Framework einsetzen können.

## 4 Erweitertes Vorgehensmodell zur Entwicklung laufzeitkritischer Software

Das einführende Beispiel zeigt, dass die systematische Umsetzung nicht-funktionaler Eigenschaften einer Software vielfältige Auswirkungen auf den Entwicklungsprozess dieser Software hat bzw. haben sollte. Ziel dieses Kapitels ist eine strukturierte Zusammenstellung von methodischen Bausteinen zur Unterstützung dieser Aspekte. Diese werden in Form von NFE-Prozessmustern als notwendige Erweiterungen von Vorgehensmodellen systematisch beschrieben. Ziel ist die Integration NFE-relevanter Konzepte aus weiteren Bereichen eines Prozessmodells, wie der Qualitätssicherung, mit dem Vorgehensmodell, um deren Einsatz innerhalb der Entwicklung einer kritischen Software sicherzustellen.

Vorangestellt ist dieser Beschreibung ein Überblick über bereits existierende Ansätze zur NFE-Unterstützung in Vorgehens- und Prozessmodellen. Auf der Grundlage dieser Analyse werden die notwendigen Erweiterungen festgelegt und diskutiert. Darauf aufbauend wird der Begriff des NFE-Prozessmusters eingeführt. Im Anschluss daran werden diese in verschiedenen NFE-Prozessmustern zusammengefasst. Die definierten NFE-Prozessmuster werden beschrieben und in den Kontext des erweiterten Vorgehensmodells für laufzeitkritische NFE eingeordnet.

### 4.1 NFE-Unterstützung in bestehenden Prozessmodellen

Um ein Prozess-Framework mit dem Fokus der Unterstützung von nicht-funktionalen Anforderungen zu entwickeln, ist es unerlässlich bestehende prozessbasierte Ansätze auf deren Integration von Techniken oder Methoden zur Umsetzung solcher Anforderungen hin zu analysieren. Dazu werden sowohl allgemeine Prozessmodelle als auch Modelle mit expliziter Ausrichtung auf die Unterstützung spezifischer nicht-funktionaler Anforderungen betrachtet.

#### 4.1.1 Allgemeine Prozessmodelle

Zur systematischen und strukturierten Entwicklung von Software wurden eine Vielzahl unterschiedlicher Softwareentwicklungsprozesse definiert. Diese reichen von leichtgewichtigen Prozessen der agilen Softwareentwicklung bis hin zu schwergewichtigen, wie dem V-Modell XT. In diesem Abschnitt sollen drei bekannte Prozessmodelle, deren Grundstruktur bereits in Kapitel 2.5 erläutert wurde, auf ihre NFE-Unterstützung hin betrachtet werden.

##### V-Modell XT

Das V-Modell XT ist ein sehr allgemeingültiges Prozessmodell, welches auf keinen spezifischen Modellierungstechniken oder Werkzeugen beruht. Es definiert mit seinen Vorgehensbausteinen relativ grobgranulare Prozessartefakte. Die Vorgehensplanung ist dabei eher dokumenten- als aktivitätsorientiert. Dies wird auch an der Definition des Vorgehensbausteins *Qualitätssicherung (QS)*, welcher auf einem bereitgestellten *QS-Handbuch* beruht, deutlich. Dieses ist wie folgt in der Spezifikation beschrieben [KBS09]:

„Das **QS-Handbuch** beinhaltet eine Kurzbeschreibung der Qualitätsziele im Projekt, die Festlegung der zu prüfenden QS-Produkte und Prozesse, die Organisation und Vorgaben für die Planung und Durchführung der Qualitätssicherung im Projekt sowie die Vorgaben für die Qualitätssicherung von externen Zulieferungen. Der QS-Verantwortliche muss dieses zentrale Produkt in Abstimmung mit den Schlüsselpersonen des Projekts erarbeiten. Dabei werden im QS-Handbuch insbesondere auch Häufigkeit und Notwendigkeit der Erzeugung weiterführender Produkte, die für die Qualitätssicherung im Projekt notwendig sind, festgelegt, zum Beispiel QS-Berichte, QS-Nachweisakten und Prüfprotokolle.“

In der gesamten Spezifikation des V-Modells XT finden sich nur wenige konkrete Ansatzpunkte für die Prozessunterstützung nicht-funktionaler Systemeigenschaften. In den Vorgehensbausteinen *Anforderungen* und *Systemspezifikation* werden nicht-funktionale Anforderungen zwar erwähnt, aber keine konkreten Maßnahmen zu deren Behandlung definiert. Außerdem wurde in der aktuellen Version 1.3. des V-Modells der Vorgehensbaustein *Sicherheit* überarbeitet und an die Anforderungen aus Behörden und Bundeswehr angepasst. Dabei wurden die Produkte IT-Sicherheitskonzept, Datenschutzkonzept und Funktionssicherheitskonzept eingeführt. Dabei beschränkt sich das V-Modell, wie in allen Vorgehensbausteinen, auf eine abstrakte Beschreibung der Aktivitäten und Produkte.

Im Teil 8 der Spezifikation ist zwar eine lose Zusammenstellung von sogenannten Methodenreferenzen enthalten. Diese beschreiben allerdings nur allgemein bekannte Verfahren der Softwartechnik und keine Verfahren zur konkreten Unterstützung nicht-funktionaler Laufeigenschaften.

##### Rational Unified Process

Der Rational Unified Process (RUP) definiert einen Arbeitsablauf (Workflow) *Test*. Dieser zielt in erster Linie auf die Messung der Produktqualität im Hinblick auf das Zusammenspiel der Komponenten, Integrationsaspekte, Funktionalität und Performanz ab. Die Art und Weise der Umsetzung der Tests ist im RUP nicht explizit vorgeschrieben, sondern kann durch Best Practice Methoden erfüllt werden. Was fehlt ist ein systematisch definiertes Vorgehen zur gesamtheitlichen Qualitätssicherung [Hes00].

Der RUP definiert die Qualität eines Produktes sowohl durch die Qualität der erstellten Software selbst als auch durch die Qualität des Prozesses. Es existiert aber keine übergeordnete Rolle *Qualitätsingenieur*. Die Erfüllung der Qualität liegt grundsätzlich in der Verantwortlichkeit von jedem Einzelnen. Das heißt,

„Jeder Projektbeteiligte teilt die Verantwortung und die Ehre, wenn sie erreicht wird (aber auch die Schande, wenn nicht).“ [Kru03]

Die nicht-funktionalen Eigenschaften stehen also nicht im Mittelpunkt des RUP. Dennoch integriert der RUP einige Konzepte, die die allgemeine Qualitätsicherung positiv beeinflussen. Dazu gehören neben dem Testworkflow, die starke Integration der Anforderungsanalyse in der „Inception Phase“, die architekturgetriebene sowie die iterative Entwicklung der Software. Außerdem unterstützt der RUP durch den unterstützenden Workflow des „Configuration and Change Managements“ die Verfolgbarkeit der Anforderung [ZHG05].

### Agile Prozessmodelle

Agile Ansätze fokussieren auf eine starke Integration des Kunden in die Entwicklung, favorisieren den „Test First“-Ansatz und minimieren den Modellierungs- und Dokumentationsaufwand. Durch die kurzen Iterationsphasen bei der Entwicklung entstehen frühzeitig Prototypen, die neben den funktionalen auch nicht-funktionale Anforderungen umsetzen können. Damit sind die agilen Prozessmodelle für kleine Projekte sehr gut geeignet, um Qualität im Allgemeinen zu unterstützen. Bei großen Projekten kann XP jedoch aufgrund der fehlenden Dokumentationen und damit unzureichender Kommunikationsmittel zu organisatorischen Schwierigkeiten führen ([ZHG05], [Bor04]).

### Bewertung

In bekannten Prozessmodellen stehen nicht-funktionale Eigenschaften nicht im Mittelpunkt der Betrachtung. Daher werden qualitätssichernde Maßnahmen nur im Allgemeinen gefordert. Deren konkrete Umsetzung ist nicht Inhalt der Prozessmodelle. Dies hat den Vorteil, dass die Prozessmodelle für alle nicht-funktionalen Eigenschaften anwendbar sind. Allgemeingültige qualitätssichernde Maßnahmen sind beispielsweise die iterative Entwicklung, das Anforderungsmanagement und durchgängige Qualitätssicherung im Allgemeinen. Außerdem ist zu erkennen, dass agile Prozesse für kritische Systeme unzureichend sind, da sie entsprechende Maßnahmen nur unzureichend fordern.

Nachteilig ist, dass sich die an der Ausführung des Prozesses innerhalb eines Projekts beteiligten Mitarbeiter, das Wissen über konkrete Techniken zur Umsetzung der Vorgehen aus externen Quellen beziehen müssen. Dies kann zu erheblichen Inkonsistenzen bei der Behandlung dieser Eigenschaften führen.

### 4.1.2 NFE-spezifische Prozessmodelle

Da die klassischen Prozessmodelle die Qualitätseigenschaften nicht im Fokus ihres Prozesses haben, wurden verschiedene andere Vorgehen entwickelt, um die konkrete Umsetzung solcher Eigenschaften zu unterstützen. Zwei der am weitesten entwickelten werden im Folgenden kurz vorgestellt.

#### Software Performance Engineering

Software Performance Engineering (SPE) wurde erstmals in [Smi90] erwähnt und führt ein systematisches Prozessmodell ein, dass es ermöglicht, die Performanz von Software auf verschiedenen Stufen der Softwareentwicklung zu berechnen. Abbildung 4.1 stellt

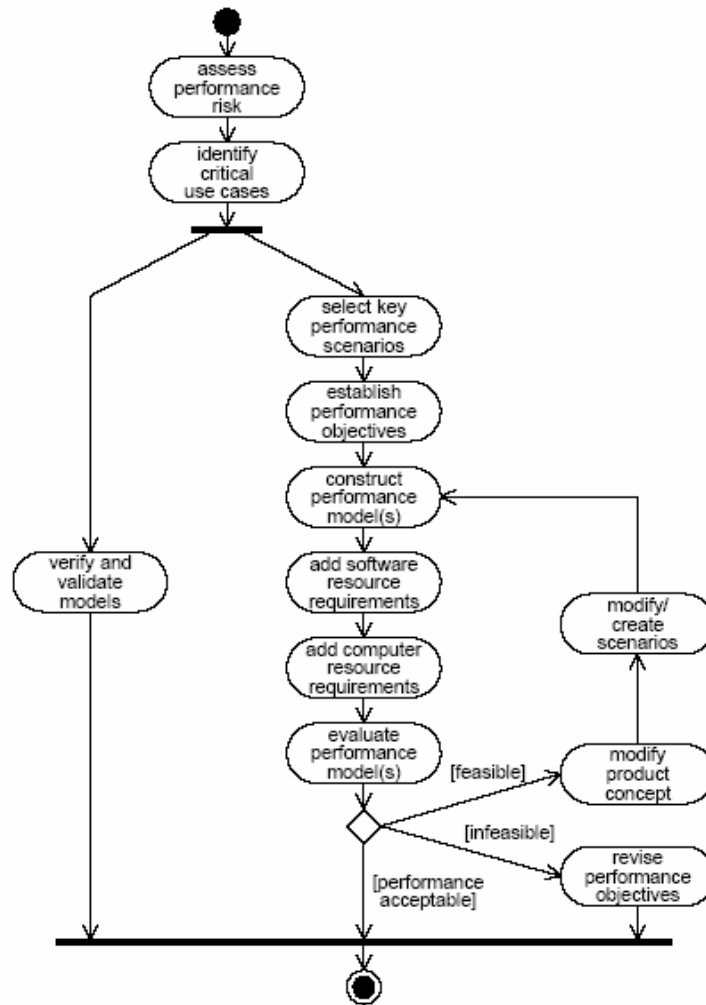


Abbildung 4.1: SPE Prozess (aus [SG02])

den Prozessablauf schematisch dar. Er basiert auf einer sorgfältigen und methodischen Bewertung der Performanz innerhalb des Lebenszyklus von den Anforderungen über die Modellierung bis zur Implementierung und Wartung. Dabei wird die Performanzanalyse immer bezüglich der kritischen Szenarien durchgeführt. Der Prozess besteht aus zwei parallelen Subprozessen: dem Analysepfad der Verifikation und Validierung und dem Modellierungspfad. Das Basiskonzept von SPE ist die Trennung in die Beschreibung der Software in *Software Execution Models* und der Beschreibung der Umgebung in *System Execution Models*. Dadurch wird eine zweistufige Analyse ermöglicht. Auf der ersten Stufe, den Softwaremodellen, werden statische Antwortzeitanalysen durchgeführt. Dadurch sind Defizite bei Architekturentscheidungen bezüglich der Performanz aufdeckbar. Auf der zweiten Stufe werden in dynamischen Umgebungsmodellen zusätzliche Einflussfaktoren der Umgebung modelliert, beispielsweise Systemauslastungen oder mehrere Nutzer.



Analysen dieser Modelle liefern Ergebnisse, wie die Identifikation kritischer Teile der Architektur, von Ressourcenengpässen oder Informationen über die Skalierbarkeit der Architektur [SW03].

Obwohl sich SPE nur mit einem sehr spezifischen Problemraum beschäftigt, sind dessen folgende Eigenschaften für einen Entwicklungsprozess für lauffzeitkritische Systeme interessant:

- Beginn der Entwicklung mit dem kritischsten Szenario
- Trennung der Modelle in zwei Abstraktionsstufen und dadurch Einsatz verschiedener (sich ergänzender) Analysen
- Iterative Entwicklung von Systemen

Ein Vorteil der Trennung der beiden Modellstufen ist die Möglichkeit der Wiederverwendung der Modelle in anderen Entwicklungen. So können die Softwaremodelle mit verschiedenen Umgebungsmodellen kombiniert werden und umgekehrt. Außerdem wird zuerst das „einfachere“ Softwaremodell analysiert, wodurch eventuelle Modellierungsfehler eher im Entwicklungszyklus gefunden werden.

Einen umfassenden Überblick über Entwicklungen, welche auf dem Modell von SPE basieren, gibt [BMIS03]. Die selben Autoren haben sich in [BS04] auch mit der Einordnung der Modellierung von Performanz in Softwareentwicklungsprozesse beschäftigt. Das Ergebnis ist ein ähnliches Prozessmodell, wie das in Abbildung 4.1 bereits dargestellte. Es wird zudem festgestellt, dass die Art der Modellierung der Performanz keinen Einfluss auf die Anordnung der Prozessschritte hat.

#### Prozess Modell für „Security Engineering“

Mehrere Arbeiten der „Quality Engineering“-Gruppe der Universität Innsbruck beschäftigen sich mit der Spezifikation eines Prozessmodells für Security Engineering (z.B. [BBH<sup>+</sup>03], [BBHP04]). Sie definieren Sicherheit als eine Eigenschaft, welche spezifische Modellierungstechniken benötigt und in allen Phasen der Entwicklung betrachtet werden muss. Da Sicherheitseigenschaften meist nur auf technischem Niveau spezifiziert werden (Nutzung von Verschlüsselungstechniken, Sicherheitsprotokollen, Logging), ist es das Ziel des Ansatzes, Abstraktionsebenen zu separieren und Sicherheitsrisiken in einem geeigneten Abstraktionsgrad zu spezifizieren. Der Prozess unterscheidet zwei Abstraktionsstufen: „Application Level Models“ und „Platform-Dependent Models“. Die Gefahren, die auf der Anwendungsebene analysiert werden, bedingen dabei Gefahren die in der technischen Ebene betrachtet werden müssen.

Es wird gefordert, dass die Analyse der Sicherheitsanforderungen bereits in die Geschäftsmodellierung integriert wird, um die durchgängige Behandlung der sicherheitsspezifischen Aspekte während der Entwicklung zu unterstützen. Zur Integration der dazu notwendigen Aktivitäten der Sicherheitsanalyse wird ein Unterprozess *Security Analysis* definiert, der in den Kernprozess eingebunden wird. Abbildung 4.2 zeigt einen schematischen Überblick.

Der Unterprozess umfasst fünf Schritte [BBHP04]:

#### 4 Erweitertes Vorgehensmodell zur Entwicklung laufzeitkritischer Software

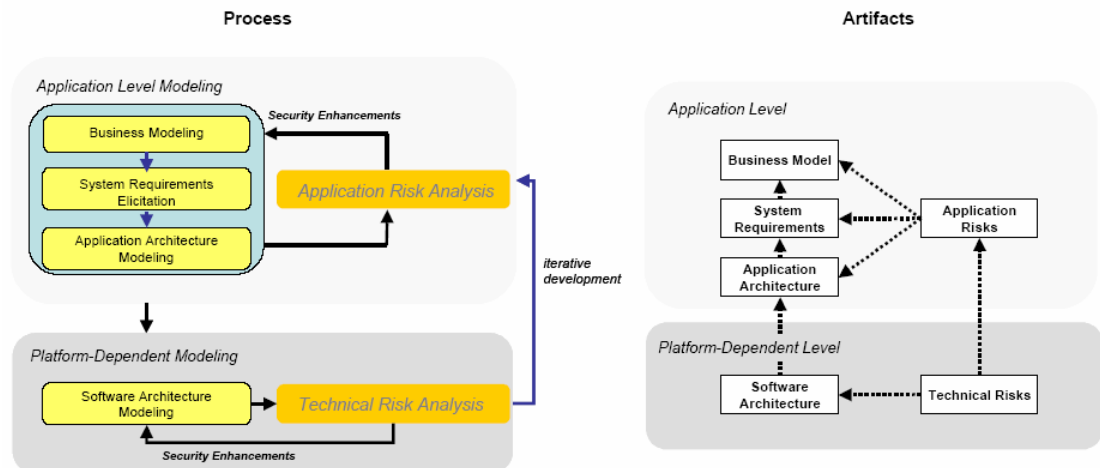


Abbildung 4.2: Entwicklung sicherheitskritischer Anwendungen (aus [BBHP04])

1. *Security Requirements Elicitation*: Spezifikation der Sicherheitsanforderungen im Kontext der Modelle des Hauptprozesses
2. *Threats Modelling*: Sammlung potentieller Gefahren in Bezug zu Sicherheitseigenschaften
3. *Risk Analysis*: Analyse der Wahrscheinlichkeit des Auftretens der Gefahren
4. *Measures Design*: Definition von Maßen für die einzelnen Risiken und Integration dieser in die Modelle des Hauptprozesses
5. *Correctness Check*: Überprüfung der Maße (formal oder informal) gegen die Anforderungen und Entscheidung. Daraus ergeben sich Sicherheitsanforderungen auf der plattform-spezifischen Ebene.

Der Unterprozess zur Analyse der Maße dieser Sicherheitsanforderungen muss in beiden Abstraktionsebenen der Modelle integriert werden. Er ist wiederholt ausführbar, da die Entwicklung der Anwendung iterativ erfolgt. Das Prozessmodell beschreibt nicht nur Aktivitäten, sondern auch die Notwendigkeit der Erweiterung von Artefakten um Sicherheitsaspekte (Abbildung 4.2, rechte Seite).

In Erweiterung dieses Ansatzes definiert Popp Teilprozesse zur Integration von Aspekten der Zugriffssicherheit in die frühen Phasen der Systementwicklung [Pop05]. Dabei wendet er den Unterprozess (in [Pop05] als Sicherheitsmikroprozess bezeichnet) auf die folgenden, von ihm definierten, Teilprozesse an: Geschäftsprozessmodellierung zugriffssicherer Systeme, Anwendungsfallmodellierung zugriffssicherer Systeme und Analyse zugriffssicherer Systeme.

Dieses Prozessmodell besitzt ebenfalls einige Eigenschaften, die für einen Entwicklungsprozess für laufzeitkritische Systeme interessant sind:

- Erweiterung der Modelle um Möglichkeiten der Spezifikation von Sicherheitsaspekten
- Trennung von Sicherheitsaspekten in zwei Abstraktionsebenen und entsprechend differenzierte Analysemöglichkeiten
- Analyse der kritischen Gefahren zuerst
- Rückverfolgbarkeit (Traceability) von Sicherheitsanforderungen über Abstraktionsebenen (Anforderungen der Anwendungsebene ergeben detailliertere Anforderungen oder erfüllte Eigenschaften der technischen Ebene)

#### **Bewertung**

Wenngleich beide Prozessmodelle auf verschiedene nicht-funktionale Eigenschaften spezialisiert sind bzw. sich mit speziellen Aspekten der Umsetzung beschäftigen, lassen sich einige allgemeingültige Eigenschaften herausfiltern. Dazu gehören vor allem die iterative Entwicklung entlang des kritischen Szenarios sowie die Integration von Analysen und Verifikationen bereits auf Modellebene, um Modellierungsfehler noch vor ihrer Umsetzung in Code und damit so früh wie möglich zu finden. Voraussetzung dafür ist die Unterscheidung unterschiedlicher Abstraktionsstufen der Modellierung. Daraus resultieren weitere wesentliche Anforderungen an den Entwicklungsprozess. Dies beinhaltet die Verfolgbarkeit (Traceability) von nicht-funktionalen Anforderungen, die Trennung von Anforderungen und Maßen (Measures), die Korrektheit der gemessenen Maße sowie die Vollständigkeit der Anforderungen und Maße.

#### **4.1.3 Weitere Techniken zur Behandlung von NFE**

Obwohl es nur wenige durchgängige Prozessbeschreibungen im Kontext nicht-funktionaler Eigenschaften gibt, haben sich viele Autoren mit der Behandlung nicht-funktionaler Eigenschaften in einzelnen Phasen der Softwareentwicklung beschäftigt. Dabei können Ansätze der Anforderungsanalyse, der Systemspezifikation bzw. -modellierung und Analysetechniken unterschieden werden. Außerdem von Interesse sind Ansätze zur Behandlung nicht-funktionaler Eigenschaften in komponentenbasierten Systemen sowie Entwurfsmuster. Die einzelnen Arbeiten behandeln meist konkrete Techniken zur Behandlung nicht-funktionaler Systemeigenschaften. Die Darstellung der einzelnen Techniken liegt dabei nicht im Fokus dieser Arbeit. Die nächsten Abschnitte fassen vielmehr die für die Definition der Vorgehenserweiterungen notwendigen Kerngedanken innerhalb der einzelnen Phasen zusammen.

#### **Anforderungsanalyse**

Das Problem nicht-funktionaler Eigenschaften in der Anforderungsanalyse ist es, dass innerhalb der Vorgehensmodelle oft nur deren informelle Spezifikation gefordert wird [Bor04]. Um die Erfüllung der Eigenschaften durch Tests und Verifikationsverfahren zu überprüfen, ist es notwendig die NFE explizit zu analysieren und zu erfassen. Eine Möglichkeit dafür sind zielorientierte Ansätze wie das bereits erwähnte NFR-Framework

[CNYM00]. Das wesentliche Konzept dieses Frameworks ist die hierarchische And/Or-Dekomposition nicht-funktionaler Eigenschaften, die als Softgoals bezeichnet werden. Diese Softgoals werden entweder mit messbaren Eigenschaftswerten belegt oder in operationalisierbare Eigenschaften überführt. Ähnliche Ansätze stellen beispielsweise [LX99], [YdPLM04] und [Hei05] vor. Letztere Arbeit beinhaltet außerdem einen Vergleich verschiedener zielorientierter Methoden der Anforderungsanalyse.

Es existieren demnach verschiedene Herangehensweisen zur Analyse und Erfassung von NFE. Diese Verfahren arbeiten nach anderen Paradigmen als die Analyse funktionaler Eigenschaften, da beispielsweise das zielorientierte Vorgehen das Wesen der NFE besser widerspiegelt als Use Case basierte Ansätze. Daher muss es einen zusätzlichen Prozessschritt geben, der die Analysen der funktionalen und nicht-funktionalen Eigenschaften verknüpft ([TT05], [CS06]). Nur so kann gewährleistet werden, dass die NFE in den Architekturentwurf einfließen.

Ein weiteres Problem zeigt sich bei der Analyse von sicherheitskritischen Systemen. Hier sind zusätzlich zu den Anforderungen auch die Bedrohungen denen das System unterliegen kann zu analysieren. Zur Spezifikation derartiger Szenarien können sogenannte Misuse Cases [SO01] eingesetzt werden.

Nicht-funktionale Eigenschaften können ebenso als Aspekte betrachtet werden. Aspekte in der Anforderungsanalyse werden als *Early Requirements* bezeichnet. Prinzipiell ist die Behandlung als Aspekt eine weitere mögliche Technik im Sinne dieser Arbeit. Als Einstieg in die Umsetzung der notwendigen Anforderungsanalyse der NFE mittels Aspekten kann [Mor07] empfohlen werden.

### Systemspezifikation und -modellierung

Zur Spezifikation nicht-funktionaler Eigenschaften existiert eine Vielzahl von Techniken. Die OMG definiert beispielsweise verschiedene UML-Profile, wie das „UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms“ [Obj08b] oder das „UML Profile for Schedulability, Performance and Time“ [Obj05]. Parallel dazu gibt es eine Vielzahl nicht standardisierter Profile, wie beispielsweise UMLsec [Jür05] für die Modellierung und Spezifikation von Sicherheitsaspekten. Der Einsatz der verschiedenen UML-Profile ist spezifisch auf die jeweilig behandelten nicht-funktionalen Eigenschaften spezialisiert. Daher gibt es Bestrebungen nach generischen Beschreibungstechniken, die verschiedene Ausprägungen nicht-funktionaler Eigenschaften abdecken. [RZ03] und [Zsc09] sind Beispiele für solche Beschreibungstechniken auf der Basis von Charakteristiken und Maßen.

Die Liste der Spezifikationstechniken für nicht-funktionale Eigenschaften kann beliebig erweitert werden. Einen detaillierten Überblick, sowie eine Klassifikation verschiedener Ansätze bieten ([Zsc07], S.128ff) und [Zsc]. Die einzelnen Ansätze müssen nicht im Detail dargestellt werden, um zu sehen, dass diese ein deutliches Beispiel für die bereits diskutierte Vielfalt der Umsetzungsmöglichkeiten von NFE-spezifischen Aktivitäten sind. Innerhalb der Makroprozesse ist demnach nur eine entsprechende Aktivität zu integrieren. Dazu sollte das Wissen über geeignete Techniken der Systemspezifikation- und modellierung den Projektbeteiligten auf geeignete Weise zur Verfügung stehen.

### Analyse- und Entwurfsmuster

Der Einsatz von Analyse- und Entwurfsmustern hat positive Auswirkungen auf die Qualität einer Software, da bewährtes Wissen angewendet wird. Allerdings gewähren die gängigen Muster der Softwaretechnik nicht zwangsläufig lauffähige bzw. harte nicht-funktionale Anforderungen. Daher gibt es einige Arbeiten, die sich mit der Beschreibung von Analyse- und Entwurfsmustern für die Sicherstellung solcher Eigenschaften beschäftigen.

Einen Ansatz für Analysemuster bieten verschiedenen Erweiterungen des bereits erwähnten NFR-Frameworks. Es existieren sowohl Ansätze Softgoal-Graphen mit Entwurfsmustern zu verknüpfen [GY01], als auch Softgoal-Graphen ontologiebasiert für die Entwickler nutzbar zu machen [SJP<sup>+</sup>07]. Mit Entwurfsmustern von Realzeitsystemen haben sich vor allem B. Douglass und B. Selic beschäftigt. Einen guten Überblick bieten ihre Artikel in [LMS03]. Außerdem wird in [Dou98] ein Vorgehen zur Entwicklung von eingebetteten Systemen mit dem Fokus auf Realzeitaspekte vorgestellt. Das Design erfolgt unter Verwendung zahlreicher Muster, deren Anwendung detailliert beschrieben wird. Für den Einsatz von Mustern im sicherheitskritischen Bereich sind in [SFBH<sup>+</sup>05] und [Sec] eine Sammlung von Mustern beschrieben.

Die Ausprägung der Muster selbst hat keinen Einfluss auf die Definition eines Vorgehensmodells. Ein entsprechendes Vorgehensmodell sollte Aktivitäten integrieren, die die Entwickler auf die mögliche Anwendung von Mustern hinweist und entsprechende Quellen zur Verfügung stellt.

### Techniken zur Überprüfung nicht-funktionaler Eigenschaften

Beschäftigt man sich mit der Umsetzung nicht-funktionaler Anforderungen, kommt man an Analysetechniken und Testverfahren nicht vorbei. Zur Gewährleistung der entsprechenden NFE muss deren Erfüllung kontinuierlich überprüft werden. Dies kann durch verschiedene Techniken (Testen, Verifizieren oder Simulieren) umgesetzt werden. Dabei gibt es eine ganze Bandbreite von Ansätzen, welche nicht im Detail beschrieben werden, da einzelne Techniken für die Definition des erweiterten Vorgehensmodells nicht relevant sind. Vielmehr sind entsprechende Prozessschritte der Verifikation und Validierung von NFE zu integrieren.

Im Bereich kritischer Anwendungen werden zunehmend formale Verfahren der Verifikation eingesetzt. Dadurch ist es möglich, die Erfüllung der Eigenschaften zu beweisen. Für funktionale Eigenschaften von Software ist die Anwendung formaler Verifikationstechniken sehr gut untersucht. Die systematische Verifikation nicht-funktionaler Eigenschaften wird dagegen in Forschungsarbeiten diskutiert, ist aber nicht allgemeiner Stand der Technik.

Ein interessanter Ansatz zur Anwendung der modellgetriebenen Entwicklung auf Verifikationsplattformen wird im Projekt HIDOORS [Hun04] unter dem Namen *Quality Driven Design* vorgestellt. Wie Abbildung 4.3 zeigt, werden als plattformspezifisches Modell (PSM) nicht nur Plattformen zur Ausführung von Software (z.B. Java) sondern auch Plattformen zur Verifikation abstrahiert. Durch die Bereitstellung einer MDA-ähnlichen Architektur ist es so möglich das PIM auch in verschiedenartige Verifikationsmodelle zu transformieren und entsprechende Analysen kritischer Systemeigenschaften durch-

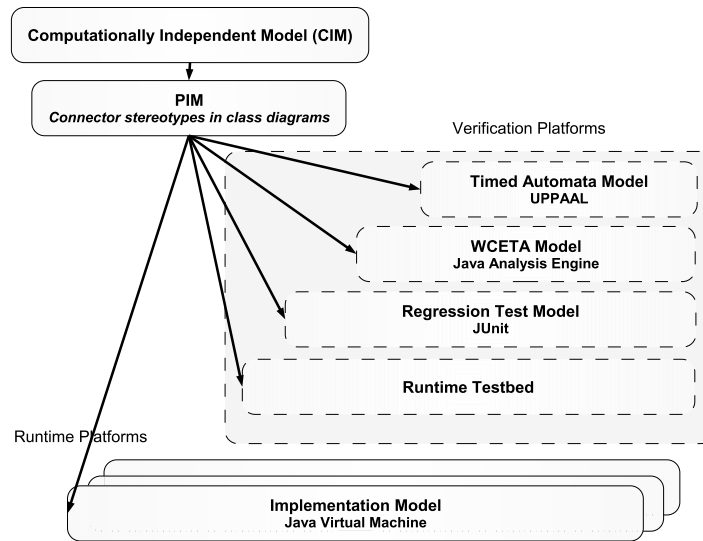


Abbildung 4.3: Parallele Verifikationsplattformen (aus [HAH05])

zuführen. Mit der Integration von Verifikationstechniken in die MDA haben sich noch andere Arbeiten beschäftigt, welche sich unter dem Begriff *modellgetriebene Verifikation* (Model Driven Verification, MDV) zusammenfassen lassen. Dazu zählen: [SE03], [SPGM05] und [Bar03].

### Bewertung

Dieser Überblick zeigt deutlich, dass in verschiedensten Phasen der Entwicklung unterschiedlichste Techniken notwendig sind, um nicht-funktionale Anforderungen an ein System kontinuierlich zu behandeln und damit systematisch umzusetzen. Dabei ist eine große Bandbreite und Vielschichtigkeit der Lösungen zu erkennen. Die meisten Ansätze sind auf der Mikroebene der Prozessbeschreibungen einzuordnen, da sie sich mit der konkreten Umsetzung beschäftigen. Es erfolgt in den Arbeiten jedoch keine methodische Integration in Vorgehensmodelle.

#### 4.1.4 Zusammenfassende Bewertung der Unterstützung

Der vorangestellte Überblick zeigt, dass es bereits eine Vielzahl von Ansätzen auf dem Gebiet der Prozessunterstützung für nicht-funktionale Systemeigenschaften gibt. Die nähere Analyse ergibt, dass dies bisher nur sehr unsystematisch erfolgte. Das liegt zum Einen daran, dass die Beschäftigung mit der Problematik der NFE noch sehr jung ist. Zum Anderen kommt auch hier wieder die Schwierigkeit der Komplexität und Vielschichtigkeit nicht-funktionaler Eigenschaften zum Tragen. So existieren derzeit entweder Prozessmodelle für eine Klasse nicht-funktionaler Eigenschaften oder aber es werden einzelne Verfahren bzw. Techniken bereitgestellt, die zwar mehrere NFE abdecken aber keinen durchgängigen Entwicklungsprozess beschreiben.

Der derzeitige Stand der Technik weist folgende Defizite und Lücken in der kontinuierlichen Prozessunterstützung von NFE-Aspekten auf:

- Es existieren Prozessbeschreibungen auf zwei Abstraktionsebenen (Makro- und Mikroebene).
- Es gibt derzeit keine Unterstützung zur Verknüpfung der Ebenen.
- Es werden entweder einzelne Entwicklungsschritte oder einzelne NFE abgedeckt.
- NFE werden oft nur informell in der Anforderungsanalyse erfasst.
- Der Übergang von Anforderungsanalyse zu Entwurf ist unzureichend untersucht. Existierende Vorgehen sind meist durch Szenarien getrieben (use case driven), NFE-Analysen sind dagegen zielorientiert (goal driven).
- Die geforderte Integration von Verifikation und Validierung ist unzureichend definiert.

Viele Ansätze beschreiben ähnliche Mittel und somit können verallgemeinerbare Methoden herausgefiltert werden, die Bestandteile eines Vorgehensmodells zur Entwicklung laufzeitkritischer Systeme sein sollten. Damit beschäftigt sich der folgende Abschnitt. Jedoch lassen sich nicht alle aufgezeigten Methoden als generische Aufgaben in einen Prozess integrieren. Der Einsatz von Entwurfsmustern ist ein solches Beispiel, da die Entwurfsmuster an sich keinen Einfluss auf den Prozessablauf haben. Dennoch sollte das Nachdenken über den Einsatz geeigneter Entwurfsmustern eine Aktivität innerhalb des Prozessmodells sein. Die Entwurfsmusterbeschreibungen stehen dabei als Lösungsmittel zur Verfügung.

## 4.2 Notwendige Eigenschaften des erweiterten Vorgehensmodells

Aus den vorgestellten Ansätzen können notwendige Eigenschaften von Vorgehensmodellen mit systematischer NFE-Unterstützung gefolgert werden. Darüber hinaus haben sich verschiedene Autoren mit allgemeinen Anforderungen an die Entwicklung kritischer Systeme beschäftigt. Eine sehr gute Zusammenfassung dieser allgemeinen Kriterien bietet [Som07]. Sommerville definiert drei einander ergänzende Kriterien, die bei der Entwicklung kritischer Systeme zu erfüllen sind: Fehlervermeidung, Fehlerentdeckung und Fehlertoleranz. Prozesse, die auf die Erfüllung dieser Kriterien ausgerichtet sind, bezeichnet er als *Verlässliche Prozesse*. Aus den von ihm definierten Eigenschaften verlässlicher Prozesse lassen sich die ersten allgemeingültigen Eigenschaften des Vorgehensmodells ableiten:

Das Vorgehensmodell sollte folgende allgemeine Eigenschaften erfüllen:

- (0.1) Dokumentierbarkeit
- (0.2) Standardisierung des Vorgehens
- (0.3) Auditierbarkeit
- (0.4) Vielfältige und redundante Verifizierungs- und Validierungsaktivitäten
- (0.5) Robustheit in Bezug auf das Scheitern einzelner Prozessaktivitäten

Besitzt ein Vorgehensmodell diese Eigenschaften ist auch die Forderung der Prozessbewertungsmethoden, wie CMMI, nach einem definierten Prozessmodell und der formalen Prüfung der Einhaltung der Vorgaben für das Prozessmodell und dessen Artefakte erfüllt. Diese Forderungen werden bereits auf Stufe 2 des Reifegradmodells gefordert.

Eine weitere Eigenschaft, die sich aus den bereits vorhandenen Arbeiten ergibt, ist die Unterstützung eines modellbasierten Entwicklungsprozesses. Durch den Einsatz formal definierter Modellierungssprachen und automatisierter Transformationen in einem modellbasierten Vorgehensmodell kann die Softwarequalität gesteigert werden. Außerdem ist die bessere Handhabbarkeit von Komplexität durch Abstraktion ein weiterer Vorteil [SV05].

In den folgenden Absätzen werden nun die phasenspezifischen Eigenschaften dargestellt. Dabei handelt es sich um Eigenschaften, die für alle NFE gültig sind. Sie sind anhand der allgemeingültigen Projektphasen Anforderungsanalyse, Architektur (Grobentwurf), Design (Feinentwurf), Implementierung und laufendes System geordnet. Die einzelnen Phasen werden in der Literatur teilweise unterschiedlich bezeichnet, umfassen aber im Allgemeinen eine ähnliche Aufgabenstruktur. Zusätzlich werden noch Eigenschaften definiert, die nicht einer konkreten Phase zuzuordnen sind. Diese werden als phasenübergreifend bezeichnet.

##### 4.2.1 Phasenübergreifende Eigenschaften

Folgende Eigenschaften sind nicht eindeutig einer Phase zuzuordnen. Sie sind vielmehr während des gesamten Entwicklungszyklus gültig.

###### Explizite Spezifikation

In der Literatur kristallisiert sich folgende Hauptaktivität bezüglich nicht-funktionaler Eigenschaften heraus:

- (1.1) Explizite Erfassung und Spezifikation der nicht-funktionalen Anforderungen durch Angabe eines konkreten Maßes

Die Erfassung der Anforderungen erfolgt natürlich in der Anforderungsanalyse. Folgende Eigenarten nicht-funktionaler Eigenschaften erschweren dies [KS98]:

- Verschiedene Bedingungen sind hochgradig subjektiv und nur durch eine komplexe



empirische Evaluierung zu determinieren, z.B. „Das Bremssystem muss schnell reagieren.“.

- NFE beziehen sich oft auf mehr als eine funktionale Anforderung, so bezieht sich die Integrität von Daten in einem Online-System sowohl auf die Datenhaltung als auch den Datentransfer über ein Netzwerkprotokoll.
- NFE beeinflussen bzw. widersprechen sich eventuell gegenseitig, so verlangsamen beispielsweise Sicherheitsprotokolle die Systemantwortzeit
- Es existieren keine universellen Regeln und Richtlinien, um zu definieren, wann NFE optimal erfüllt sind.

Da sich diese Arbeit vorerst auf die Behandlung quantitativer nicht-funktionaler Laufzeiteigenschaften beschränkt, sind diese Schwierigkeiten durch die Angabe eines konkreten Maßes der Anforderung lösbar. Damit wird deren Erfüllung objektiv messbar. Außerdem kann über die Höhe eines konkreten Wertes besser verhandelt und entsprechende Risiken abgeschätzt werden.

Interessant für die Definition der Prozesseigenschaften wären auch Studien über die Problematik nicht-funktionaler Eigenschaften in realen Projekten. Leider sind diese nur sehr selten öffentlich zugänglich. [Bor04] führten z.B. eine derartige Studie in zwei Unternehmen durch. Dabei wurde festgestellt, dass in großen Unternehmen die Kommunikation der NFE problematisch ist. Daher ist die explizite Spezifikation und damit Dokumentation dieser Eigenschaften in allen Phasen der Entwicklung notwendig.

### Anforderungsmanagement

Das Management der nicht-funktionalen Anforderungen ist von genauso hoher Bedeutung, wie das der funktionalen Anforderungen. Daher gelten hier die gleichen Kriterien. Zusätzlich dazu muss bei der Verfolgbarkeit der NFE beachtet werden, dass diese sich über die Systementwicklung hin verändern. Dazu zählen sowohl die Dekomposition des Wertes der Anforderung auf mehrere funktionale Komponenten als auch die Aufspaltung einer allgemeineren Eigenschaft in mehrere Unterklassen oder das Abbilden auf Funktionalität. Letzteres wird oft als Operationalisierung bezeichnet. Um diese Entwicklungen nachvollziehbar zu machen ist die

(1.2) Unterstützung der Rückverfolgbarkeit (Traceability) der NFE von den Anforderungen bis zum Code

notwendig.

### Zusammenspiel funktionaler und nicht-funktionaler Anforderungen

Ein weiteres wichtiges Kriterium ist das Zusammenspiel von nicht-funktionalen Anforderungen und Funktionalität, sowie die Beachtung negativer Korrelationen verschiedener nicht-funktionaler Eigenschaften. Wie bereits mehrfach diskutiert, können sich diese gegenseitig beeinflussen. Handelt es sich dabei um widersprüchliche Anforderungen, stellen diese ein Projektrisiko dar, welches klar zu bestimmen ist und wofür Lösungsstrategien

zu entwickeln sind. Daher ist in diesem Bereich folgende Eigenschaft zu erfüllen:

- (1.3) Die gegenseitige Beeinflussung verschiedener NFE sowie deren Einfluss auf die Funktionalität ist durch den Einsatz geeigneter Methoden zu analysieren. Dazu zählen:
- Ziel-Analysen
  - Trade- off Analysen
  - Priorisierung der NFE (z.B. in: obligatorisch/ optional)
  - Aufwandsschätzung
  - Kostenanalysen

Der Schwerpunkt der Umsetzung dieser Eigenschaften wird naturgemäß in der Anforderungsanalyse liegen. Es ist aber unabdingbar, diese über den gesamten Entwicklungszeitraum zu gewährleisten. Ebenso müssen bereits von der ersten Phase an entsprechende Maßnahmen in das Vorgehensmodell integriert werden, um die Durchgängigkeit der nicht-funktionalen Eigenschaften zu gewährleisten. Durch veränderte Randbedingungen sind Entscheidungen in den späteren Phasen gegebenenfalls neu zu überdenken und eventuell neue Analysen zur Aushandlung von Konflikten nicht-funktionaler Anforderungen durchzuführen.

#### Kontinuierliches Testen und Verifizieren

Eine unabdingbare Eigenschaft von qualitätsorientierten Vorgehensmodellen ist die Integration von Tests und Verifikationen. In Bezug auf die Integration von Modellverifikationen ist dabei aber auch eine Kostenanalyse notwendig, da diese beim derzeitigen Stand der Technik für größere Modelle noch sehr rechenzeitintensiv sein können. In Bezug auf die nicht-funktionalen Systemeigenschaften müssen folgende Eigenschaften erfüllt werden:

- (1.4) Kontinuierliches Überprüfen der konkreten Maße der NFE
- (1.5) Überwachung der Erfüllung globaler nicht-funktionaler Anforderungen an mehrere Komponenten mit lokalen Eigenschaften

#### Kritischer Pfad der Entwicklung

Prozesse, die sich bereits mit der Unterstützung nicht-funktionaler Eigenschaften beschäftigen, definieren die Entwicklung entlang des *Kritischen Pfades*. Dies bedeutet, dass die kritischen Bereiche einer Software vor allen anderen Bereichen entwickelt werden, um die Erfüllung der Eigenschaften zu gewährleisten. Im Anschluss daran wird inkrementell weitere Funktionalität zu diesem Kern hinzugefügt. Dadurch wird es möglich, den Einfluss der zusätzlichen Funktionalität auf die kritischen Eigenschaften, wie z.B. Performanzeigenschaften, zu überwachen. Daher muss der Prozess noch folgende phasenübergreifende Eigenschaften erfüllen:

- (1.6) Entwicklung entlang des kritischen Szenarios
- (1.7) Inkrementelle Entwicklung der kritischen Systembereiche

#### 4.2.2 Phasenspezifische Eigenschaften

Folgende Eigenschaften können konkreten Phasen der Softwareentwicklung zugeordnet werden.

##### Anforderungsanalyse

Das Ziel der Anforderungsanalyse ist es, basierend auf Gesprächen mit allen Stakeholdern, ein Anforderungsmodell bzw. -dokument zu erstellen. Diese Anforderungsspezifikation beinhaltet die Beschreibung des zu entwickelnden Systems. Es definiert WAS realisiert werden soll, ohne das WIE einer konkreten Implementierung vorweg zu nehmen [ZGK04]. Als gängiges Mittel der Anforderungsanalyse hat sich der Anwendungsfall (Use Case) etabliert. In Bezug auf die systematische Umsetzung nicht-funktionaler Anforderungen sind dabei folgende Aspekte zu beachten:

- (2.1) Finden, Validieren und explizites Repräsentieren der nicht-funktionalen Anforderungen
- (2.2) Spezifikation des expliziten Bezugs zu funktionalen Anforderungen
- (2.3) Gewährleistung der Testbarkeit der NFE. Dies ist der Fall, wenn sie quantifiziert und damit messbar gemacht worden sind.
- (2.4) Analyse der Operationalisierbarkeit der NFE und damit Abgrenzung gegen die quantitativen Maße einer Funktionalität

Ansätze zur Integration nicht-funktionaler Eigenschaften wurden bereits einige vorgestellt. Diese führen auch Analysen zu Interaktionen bzw. zu Konflikten zwischen Eigenschaften durch.

Die Anforderungsanalyse der NFE erfolgt oft zielorientiert und nicht szenarienbasiert, wie die funktionale Systemanalyse. Da die weiteren Entwicklungsschritte ebenfalls auf Szenarien basieren, ist die explizite Verknüpfung zwischen nicht-funktionalen Anforderungen und Funktionalität von enormer Wichtigkeit.

Die Anforderungsanalyse ist durch geeignete Werkzeuge zu unterstützen. Dabei werden nicht-funktionale Anforderungen unterschiedlich stark unterstützt. Für einen umfangreichen Überblick über die Mächtigkeit verschiedener Werkzeuge sei [INC] empfohlen.

##### Architekturentwurf

Der Entwurf einer Softwarearchitektur unterstützt den Übergang von den Anforderungen zur Realisierung. Ziel ist es, eine grundlegende Struktur des Systems in Form einer Systemarchitektur zu erstellen. Gleichzeitig dient sie dazu, die prinzipielle Machbarkeit des Systems zu zeigen [ZGK04]. Dabei steht die Entwicklung einer Architektur am Be-

ginn des Entwurfsprozesses und hat weitreichende Konsequenzen für das gesamte spätere Produkt.

Eine Softwarearchitektur beschreibt die Struktur des Systems, d.h. die Subsysteme bzw. die Komponenten des Systems und deren Beziehungen untereinander. Es werden sowohl statische als auch dynamische Aspekte dargestellt. Die einzelnen Komponenten sollen über Schnittstellen und abstrakte Klassen verknüpft werden, so dass die Abhängigkeiten klar definiert sind und die Komponenten austauschbar bleiben. Für eine sinnvolle Zerlegung können neben den Aspekten wie Kohäsion und Kopplung sowie Wiederverwendbarkeit von Komponenten auch die Arbeitsteilung unter den Mitarbeitern bedacht werden, die das System implementieren sollen.

Die Qualität eines Software-Systems ist wesentlich von der Wahl der Architektur abhängig. Daher sind auch die nicht-funktionalen Eigenschaften explizit in die Entwurfsentscheidungen einzubeziehen. Es ergeben sich folgende Eigenschaften:

- (3.1) Spezifikationsverpflichtung verbunden mit der Auswahl geeigneter Modellierungstechniken für NFE
- (3.2) Identifikation globaler und lokaler nicht-funktionaler Anforderungen, sowie explizite Zuordnung zu Schnittstellen durch Dekomposition
- (3.3) Systematische Präzisierung der NFE durch deren Abbildung auf funktionale Eigenschaften bzw. systemnahe nicht-funktionale Eigenschaften
- (3.4) Bildung von Arbeitspaketen als Managementunterstützung
  - Trennung NFE-kritischer Bereiche von unkritischen
  - Definition unabhängiger funktionaler Schnittstellen, die sich in ihren Qualitätsanforderungen nicht beeinflussen

Diese Eigenschaften können nicht direkt durch Werkzeuge innerhalb des Prozesses unterstützt werden, da es zwischen Anforderungsanalyse und Architektur eine Modellierungslücke gibt. Diese Lücke ist durch explizite Gedankenarbeit eines Architekten zu überbrücken. Dabei stehen ihm Entwurfsmuster als Hilfsmittel zur Verfügung.

Während die vorangegangenen Eigenschaften als Spezialisierung von (1.1.) zu sehen sind, können und müssen auch die Eigenschaften des Anforderungsmanagement (1.2), sowie des kontinuierlichen Testens und Verifizierens ((1.4) - (1.6)) wie folgt konkretisiert werden:

- (3.5) Unterstützung der Rückverfolgbarkeit der Anforderungen durch Verknüpfung der nicht-funktionalen Anforderungen an das System mit nicht-funktionalen Eigenschaften des Systems bzw. Teilen davon
- (3.6) Unterstützung von Analysen der nicht-funktionalen Maße auf Modellebene

### Feinentwurf

Der Feinentwurf berücksichtigt, im Gegensatz zum Architekturentwurf, auch alle Voraussetzungen, die sich durch den Einsatz einer konkreten Technologie ergeben. Es werden alle Teilaspekte definiert und modelliert, die für eine Implementierung des Systems notwendig sind [ZGK04]. In Bezug auf das Vorgehensmodell sind folgende Eigenschaften zu erfüllen:

- (4.1) Unterstützung von Analysen der Maße im Modell
- (4.2) Definieren von Abbildungen (Verfeinerungen) der NFE

Dabei handelt es sich wiederum um Spezialisierungen der allgemeinen Eigenschaften, hier jedoch mit dem Fokus auf den plattformspezifischen nicht-funktionalen Eigenschaften.

### Implementierung

Das Ergebnis dieser Phase sind alle Artefakte, welche zur Implementierung des Systems erstellt wurden, als Ergebnis. Dazu zählen neben dem Quellcode vor allem Subsysteme und Komponentenimplementierungen. Zur Implementierungsphase zählt ebenfalls die Systemintegration, also das technische Zusammenführen von mehreren Komponenten oder Subsystemen zu einem System [ZGK04]. Wiederum gelten folgende Eigenschaften, die sich aber nun auf Codeartefakte beziehen:

- (5.1) Unterstützung von Systemanalysen bezüglich nicht-funktionaler Maße
- (5.2) Definieren von Abbildungen (Verfeinerungen) der NFE

### Inbetriebnahme (Deployment)

Der Zweck dieser Phase ist es, das Softwaresystem an den Kunden zu übergeben. Dazu gehört neben Beta-Tests und Schulung der Endnutzer vor allem die Bereitstellung von Installationspaketen sowie Installation und Verteilung der Software.

Bezüglich der Durchsetzung der nicht-funktionalen Laufzeiteigenschaften sind dabei durch die Laufzeitumgebung Verträge zwischen den kommunizierenden Systemteilen auszuhandeln und die benötigten Betriebsmittel, wie CPU, Speicherplatz etc., zur Verfügung zu stellen. Dies kann durch Reservierung der Mittel realisiert werden.

- (6.1) Durchsetzung der NFE durch die Laufzeitumgebung

### Zusammenfassung

Ausgehend von der Analyse der bestehenden Ansätze zur Umsetzung nicht-funktionaler Laufzeiteigenschaften stellte dieser Abschnitt systematisch notwendige Eigenschaften eines erweiterten Entwicklungsprozesses zusammen. Dabei wurden sowohl phasenübergreifende als auch phasenspezifische Eigenschaften bestimmt. Diese Eigenschaften gehen in die Definition der NFE-Prozessmuster im nächsten Abschnitt ein.

### 4.3 NFE-Prozessmuster für laufzeitkritische NFE

Aus den vorab definierten notwendigen Eigenschaften der Erweiterungen von Vorgehensmodellen zur systematischen Unterstützung laufzeitkritischer Eigenschaften können *NFE-Prozessmuster* (*NFR Process Pattern*) gefolgert werden. Dieser Abschnitt definiert den Begriff des NFE-Prozessmusters, stellt ein Beschreibungsschema für derartige Muster vor und führt die definierten Muster ein. Dabei beschränkt sich dieses Kapitel auf die Definition der einzelnen Muster. Deren Einsatz innerhalb von PROKRIS-Methodik und -Framework zur Modellierung von Vorgehensmodellen für die Entwicklung laufzeitkritischer Software wird darauf aufbauend in den folgenden Kapiteln erläutert.

#### 4.3.1 Definition des Begriffs „NFE-Prozessmuster“

Die Beschreibung von Mustern ist aus der Programmierung bekannt [GHJV94]. Ähnliche Arbeiten existieren zur Beschreibung von Mustern innerhalb von Vorgehensmodellen, welche als *Process Patterns* bezeichnet werden. Die umfangreichsten Arbeiten sind dabei [Amb98] und [Amb99]. Ambler definiert Prozessmuster ausgehend von den Begriffen „Prozess“ und „Muster“. Unter einem Muster versteht man eine generelle Lösung für eine allgemeine Problemstellung, von der eine spezifische Lösung abgeleitet werden kann. Ein Prozess definiert sich durch eine Reihe von Aktionen, bei denen eine oder mehrere Eingaben verwendet werden, um eine oder mehrere Ausgaben zu erzeugen. Außerdem unterscheidet Ambler drei Arten von Prozessmustern:

*Task Process Patterns* sind detaillierte Beschreibungen einzelner Schritte, welche erforderlich sind, um bestimmte Aufgaben durchzuführen.

*Stage Process Patterns* beschreiben die Schritte, welche innerhalb einer Projektstufe iterativ durchlaufen werden. Ein Stage Process Pattern stellt eine höhere Abstraktion von Prozessmustern dar. Es besteht meist aus mehreren Task Process Patterns.

*Phase Process Patterns* beschreiben die Interaktionen zwischen den notwendigen Stage Process Patterns innerhalb einer Projektphase. Sie bestehen daher aus zwei oder mehreren Stage Process Patterns.

NFE-Prozessmuster stellen eine spezifische Art von Prozessmustern bereit. Die grundlegende Definition des Prozessmusters von Ambler wird dafür wie folgt erweitert:

**NFE-Prozessmuster** verallgemeinern Erweiterungen von Softwareentwicklungsprozessen, die sich bei der Umsetzung von nicht-funktionalen Eigenschaften als notwendig erwiesen haben. Sie beschreiben abstrahierte Aufgaben, Aktivitäten oder Phasen. NFE-Prozessmuster definieren Methoden die generisch bezüglich der spezifischen Ausprägung der nicht-funktionalen Eigenschaften sind. Die Methoden können entsprechend den geforderten spezifischen Eigenschaften durch verschiedene Verfahren bzw. Techniken umgesetzt und ausgeführt werden.

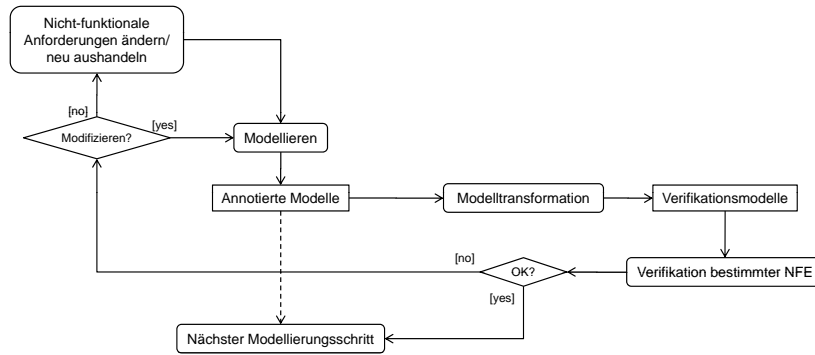


Abbildung 4.4: Aktivitätsdiagramm des NFE-Prozessmusters Verifikationsschleife

NFE-Prozessmuster	Übersetzung	Ambler
NFE-Prozessmuster für Aufgaben	NFR Task Process Pattern	Task Process Pattern
NFE-Prozessmuster für Aktivitäten	NFR Activity Process Pattern	Stage Process Pattern
NFE-Prozessmuster für Phasen	NFR Phase Process Pattern	Phase Process Pattern

Tabelle 4.1: Begriffe der Musterklassen

NFE-Prozessmuster erweitern das Vorgehensmodell und damit den Kern eines Prozessmodells. Dabei werden für NFE-relevante Konzepte aus weiteren Bereichen eines Prozessmodells, wie der Qualitätssicherung, fest mit dem Vorgehensmodell verknüpft und deren Anwendung sichergestellt.

Als Beispiel dient das NFE-Prozessmuster *Verifikationsschleife*. Anhand dieses Musters werden im Laufe der Arbeit die wesentlichen Aspekte des PROKRIS-Ansatzes demonstriert. Es beschreibt in abstrakter Sichtweise die Abfolge folgender Aktivitäten: Modellierung, Transformation in ein Verifikationsmodell, Überprüfung, Rückführung der Ergebnisse in das Originalmodell sowie Entscheidung über das weitere Vorgehen. Abbildung 4.4 zeigt die Abfolge dieser Aktivitäten.

NFE-Prozessmuster können sowohl zur Beschreibung einzelner Aufgaben als auch von Aktivitäten, bestehend aus mehreren Teilaufgaben, sowie ganzer Phasen genutzt werden. Daher ist eine Klassifizierung ähnlich der von Ambler auch für NFE-Prozessmuster sinnvoll. Zur Vereinheitlichung werden die Begrifflichkeiten an das Software Process Engineering Metamodel (SPEM, [Obj08a]) angepasst, da dieses innerhalb der Arbeit als vorwiegendes Beschreibungsmittel eingesetzt wird. Tabelle 4.1 stellt die Begriffe gegenüber.

Abschnitt	Beschreibung
Name und Klassifizierung (Name and Type)	Name des Prozessmusters und Klassifizierung
Kurzbeschreibung (Intent)	Kurze Beschreibung der Motivation, Ziele und Hintergründe des Prozessmusters
Problem (Problem)	Beschreibung des Problems bzw. der Entwicklungsaufgabe, welches vom Prozessmuster adressiert wird
Lösung (Solution)	Beschreibung des grundlegenden Lösungsansatz des Musters
Struktur (Structure)	Beschreibung des Struktur mittels einer grafischen Repräsentation der Lösung des Musters
Aktivitäten (Activities)	Beschreibung der einzelnen Aktivitäten des Musters
Produktartefakte (Work Products)	Auflistung der Ein- und Ausgabeprodukte, welche entweder vom Prozessmuster benötigt, erzeugt oder verändert werden
Kontext (Context)	Beschreibung weiterer Anforderungen, über die Produktartefakte hinaus, welche für die Ausführung des Prozessmusters gelten müssen. Vor allem werden Abhängigkeiten zwischen verschiedenen Produktartefakten sowie Bedingungen die gelten müssen, um einen Vorgang als beendet anzusehen, beschrieben.
In Beziehung stehende Muster (Related Patterns)	Auflistung von Prozessmustern, welche entweder alternativ, begleitend, im Vorfeld oder als nächste Schritte durchgeführt werden bzw. die als Teilmuster oder übergeordnete Muster ausgeführt werden können.

Tabelle 4.2: Vorlage für die Beschreibung von Prozessmustern



### 4.3.2 Beschreibungsschema für NFE-Prozessmuster

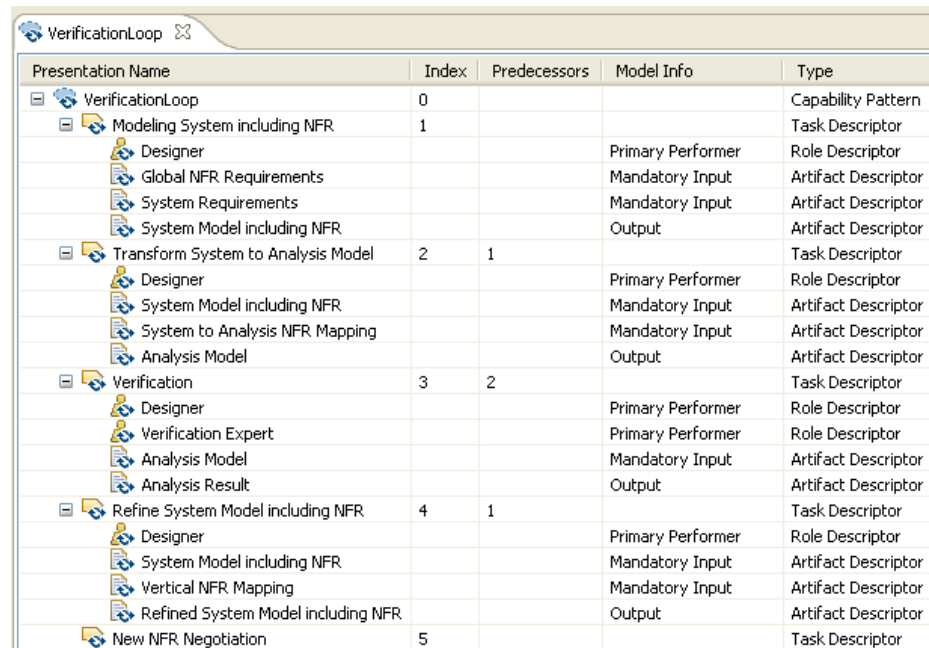
Prozessmuster müssen durch eine einheitliche Form präsentiert werden, um eine bessere Verständlichkeit und Vergleichbarkeit zu erreichen ([Rau01], [Pop05]). Dabei besteht die Beschreibung stets aus Problem, Lösung und Kontext [GHJV94]. Darauf aufbauend werden in [Rau01] und [Pop05] Mustervorlagen für die Beschreibung von Prozessmustern vorgeschlagen. Angelehnt an diese Arbeiten wurde das in Tabelle 4.2 dargestellte Beschreibungsschema entwickelt. Dieses wird zur detaillierten Beschreibung der NFE-Prozessmuster im Anhang A genutzt.

Alle Muster bestehen aus dem Trio von Problem-, Lösungs- und Kontextbeschreibung. Entscheidend dabei ist die Diskussion des Kontexts, der eine detaillierte Einordnung der Muster ermöglicht. In Abhängigkeit vom Problem und den Produktartefakten kann ein geeignetes Muster ausgewählt und die darin vorgeschlagenen Methoden in das Vorgehensmodell integriert werden.

### 4.3.3 Einsatz von NFE-Prozessmustern in der Modellierung

Die mittels des Beschreibungsschemas definierten Muster dienen Prozessexperten als Hilfestellung für die Modellierung von Vorgehensmodellen für die Entwicklung laufezeitkritischer Systeme. Es handelt sich dabei um strukturiert erfasstes Erfahrungswissen, welches auf einer mentalen Entscheidungsebene angewendet wird. Um dieses konkret in der Modellierung eines Prozessmodells zu verwenden, ist es notwendig, Prozessmusterschablonen in einer konkreten Modellierungssprache bereitzustellen. Dies ist vergleichbar mit der Unterscheidung der Ebenen von *Design Patterns*, *Design Template* und Anwendung im konkreten *Design* für Entwurfsmuster durch [Rie03]. Design Templates (Schablonen) stellen demnach Strukturen bereit, welche innerhalb eines konkreten Werkzeugs eingesetzt werden können. Dies entspricht der Umsetzung der definierten NFE-Prozessmuster durch eine Modellierungssprache innerhalb einer Prozessbibliothek. Abbildung 4.5 zeigt dies beispielhaft für das bereits eingeführte Muster Verifikationsschleife (**Verification Loop**). Zu sehen sind die Aktivitäten und deren zugeordneten Rollen und Produktartefakte. Die Modellierungssprache ist das erweiterte SPEM 2.0 und das Werkzeug die PROKRIS-Prozessbibliothek.

In der konkreten Modellierung eines Vorgehens (Design) kann das Muster (Design Template) an verschiedene Kontextinformationen angepasst werden. Dazu gehört neben den zu behandelnden nicht-funktionalen Eigenschaften auch die Phase des Entwicklungszyklus. Abbildung 4.6 stellt die Anwendung des Musters in der Modellierung eines konkreten Vorgehensmodells dar. Das NFE-Prozessmuster Verifikationsschleife ist in vier verschiedenen Ausprägungen eingesetzt: **PIM Verification**, **PSM Verification**, **Code Verification** und **Binary Verification**. Dazu sind die Aufgaben der Rolle **Verification Expert** sowie die entsprechenden Aktivitäten und Produktartefakte verfeinert worden.



Presentation Name	Index	Predecessors	Model Info	Type
VerificationLoop	0			Capability Pattern
Modeling System including NFR	1			Task Descriptor
Designer			Primary Performer	Role Descriptor
Global NFR Requirements			Mandatory Input	Artifact Descriptor
System Requirements			Mandatory Input	Artifact Descriptor
System Model including NFR			Output	Artifact Descriptor
Transform System to Analysis Model	2	1		Task Descriptor
Designer			Primary Performer	Role Descriptor
System Model including NFR			Mandatory Input	Artifact Descriptor
System to Analysis NFR Mapping			Mandatory Input	Artifact Descriptor
Analysis Model			Output	Artifact Descriptor
Verification	3	2		Task Descriptor
Designer			Primary Performer	Role Descriptor
Verification Expert			Primary Performer	Role Descriptor
Analysis Model			Mandatory Input	Artifact Descriptor
Analysis Result			Output	Artifact Descriptor
Refine System Model including NFR	4	1		Task Descriptor
Designer			Primary Performer	Role Descriptor
System Model including NFR			Mandatory Input	Artifact Descriptor
Vertical NFR Mapping			Mandatory Input	Artifact Descriptor
Refined System Model including NFR			Output	Artifact Descriptor
New NFR Negotiation	5			Task Descriptor

Abbildung 4.5: NFE-Prozessmuster Verifikationsschleife im Editor der PROKRIS-Bibliothek (Modellierung in erweitertem SPEM 2.0)

#### 4.3.4 NFE-Prozessmuster für die Anwendungsentwicklung

Aus den in Abschnitt 4.2 definierten Eigenschaften des Vorgehensmodells können verschiedene NFE-Prozessmuster identifiziert und definiert werden. Diese sind detailliert in Anhang A beschrieben. Tabelle A.1 gibt einen Überblick. Innerhalb der Entwicklung laufzeitkritischer Anwendungen und damit dem Einsatzbereich des PROKRIS-Frameworks lassen sich vier Bereiche unterteilen, in denen Erweiterungen durch NFE-Prozessmuster notwendig sind: Beschreibungstechniken für nicht-funktionale Eigenschaften, Analysetechniken, Abbildungsmechanismen und Kompositionstechniken. Bei den darin definierten NFE-Prozessmustern handelt es sich um Prozessmuster für Aktivitäten.

##### Beschreibungstechniken für nicht-funktionale Eigenschaften

Nicht-funktionale Anforderungen und Eigenschaften sind explizit zu beschreiben. Dies kann z.B. in Form von UML-Profilen geschehen oder durch formale Ansätze, wie in [Zsc07] vorgeschlagen. Dabei sind zwei verschiedene Arten von Eigenschaften zu unterscheiden:

- Eigenschaften einer Komponente oder eines Teilsystems an sich. Diese werden als *lokale* Eigenschaften bezeichnet.
- Eigenschaften die sich durch Komposition der lokalen Eigenschaften ergeben und die von außen am Gesamtsystem messbar sind. Diese werden als *globale* Eigenschaften bezeichnet.

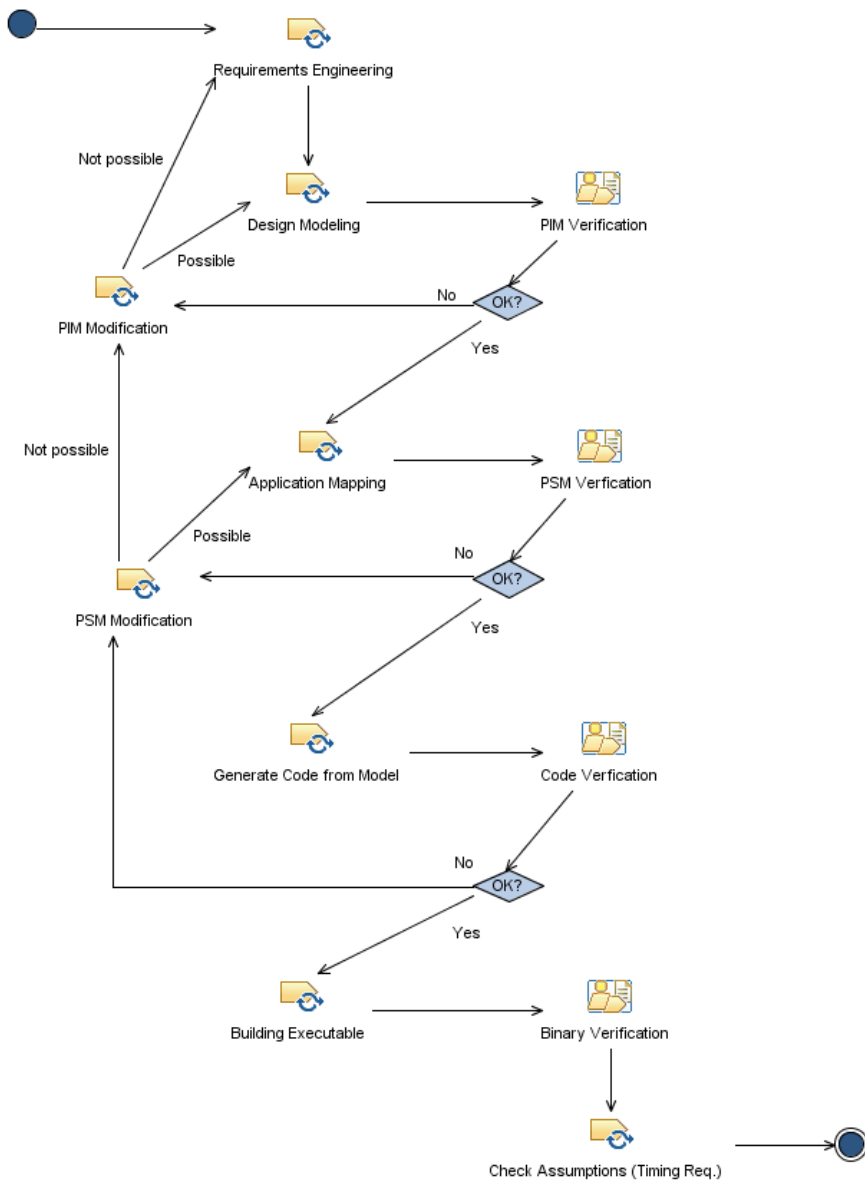


Abbildung 4.6: Anwendung von Instanzen des Prozessmusters Verifikationsschleife innerhalb eines Vorgehensmodells

In der Literatur gibt es auch andere Bezeichnungen. [Zsc07] bezeichnet diese Eigenschaften beispielsweise als extrinsische und intrinsische Eigenschaften eines Systems. Wichtig ist es, beide Arten von Eigenschaften zu spezifizieren, da im Endeffekt die Werte der globalen Eigenschaften bestimmen, ob die Anforderungen an das System erfüllt werden. Zur Spezifikation der NFE sollten Maßbeschreibungen verwendet werden [RZ04b].

**Maße** sind wiederverwendbare Spezifikationen, mit denen eine nicht-funktionale Eigenschaft in Bezug auf ein Systemmodell eindeutig formal definiert und somit messbar ist. Das System wird dabei bezüglich eines Kontextmodells definiert. Das Kontextmodell beschreibt die Konzepte des Systems, die benötigt werden, um den Wert des Maßes nachweisen zu können. Dadurch ist es möglich Maßdefinitionen unabhängig vom Anwendungssystem zu definieren und damit wiederzuverwenden. Während der Anwendungsentwicklung können die Maße mit konkreten Werten (constraints) belegt werden und stehen damit als Spezifikation nicht-funktionaler Anforderungen zur Verfügung.

Listing 4.1: Beispiel für eine Maßspezifikation unter Nutzung von CQML<sup>+</sup>

```

1  — Charakteristik zur Beschreibung einer Antwortzeit in einer Operation
   quality_characteristic response_time (op: Operation) {
3     domain: numeric real [0..) milliseconds;
     values: (op.SE->last().time() - op.SR->last().time()) abs();
5  }

7  — Diese QoS-Anweisung schränkt den Wertebereich auf 500ms ein.
   quality good_response (op: Operation) {
9     response_time (op) < 500;
   }

11 — Das QoS-Profil weist die Anweisung der Schnittstelle booking zu. Die Methode
    getCatalog() bietet (provides) die definierte Eigenschaft.
13 profile goodCatalogResponse for VideoBooking {
    provides good_response (booking.getCatalog);
15 }

```

Listing 4.1 zeigt als Beispiel eine Maßdefinition unter Nutzung der Sprache CQML<sup>+</sup> (Extended Component QoS Modeling Language, [RZ03]). Das grundlegende Konstrukt von CQML<sup>+</sup> ist die **characteristic** (Charakteristik). Sie dient der Spezifikation von nutzerspezifischen Typen (Maßen) und ist durch einen eindeutigen Wertebereich (**domain**) gekennzeichnet. Die Semantik des Maßes wird in der **values**-Klausel unter Nutzung von OCL (Object Constraint Language) definiert. OCL ermöglicht die semantische Beschreibung anhand eines in UML definierten Kontextmodells der Anwendungsdomäne. Die definierten Charakteristiken (Maßdefinitionen) sind wiederverwendbar und können durch **statements** (Anweisungen) in ihrem Wertebereich beschränkt werden. Damit erfolgt die explizite Angabe eines Wertes für das Maß. Dabei kann ein Statement sowohl eine Anforderung als auch eine Eigenschaft beschreiben. Im Beispiel von CQML<sup>+</sup> erfolgt die Zuordnung der **statements** zu einzelnen Teilen eines Systems bzw. dem Gesamtsystem durch **profiles** (Profile).

Durch die Festlegung konkreter Werte wird der Optimierbarkeit der Eigenschaften Rechnung getragen. Es gibt dadurch eine festgelegte Grenze der Optimierung. Die Eigenschaften sind in Bezug zu funktionalen Eigenschaften des Systems zu spezifizieren, da

nur so deren Erfüllung zu bewerten ist. Außerdem sind Konflikte zwischen einzelnen Anforderungen durch geeignete Techniken aufzulösen. [Poh07] unterscheidet folgende Strategien: *Verhandlung*, *Kreative Lösung* und *Entscheidung*. Zur Auflösung von Konflikten nicht-funktionaler Anforderungen werden die Strategien Verhandlung und Entscheidung empfohlen. Die Verhandlung basiert auf der Optimierbarkeit der NFE. Da es aber keine vollständige Konfliktfreiheit aller Stakeholder-Anforderungen geben kann, ist auch die Strategie der Entscheidung einzusetzen.

Folgende NFE-Prozessmuster werden in diesem Bereich definiert:

- Explizite Spezifikation der NFE
- Konfliktanalyse

Die Erweiterungen sind in allen Abstraktionsstufen der Systemmodellierung (Anforderungen, Architektur, Feinentwurf etc.) notwendig.

#### **Techniken zur Überprüfung der NFE**

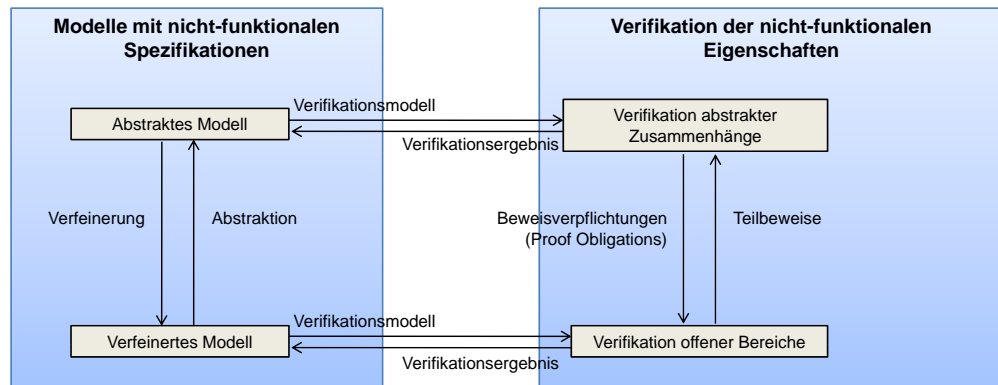
Um die Erfüllung von nicht-funktionalen Eigenschaften sicherzustellen, kommen Analysetechniken zum Einsatz. Dabei können sowohl Verifikations- als auch Simulationstechniken eingesetzt werden. Ziel ist es, die Einhaltung der Maße der nicht-funktionalen Eigenschaften zu überprüfen. In diesen Bereich fällt das bereits eingeführte NFE-Prozessmuster Verifikationsschleife. Der Umfang, den Analyseaktivitäten innerhalb eines Projekts einnehmen, ist abhängig von der Kosten- und Risikoanalyse.

Die Überprüfung der Eigenschaften sollte so früh wie möglich und damit schon auf der Modellebene erfolgen. Zu diesem Zeitpunkt der Entwicklung sind konkrete Ausführungsbedingungen (z.B. „Worst Case“-Ausführungszeiten von Methoden) noch nicht bekannt. Daher sind teilweise nur Annahmen über Werte von Eigenschaften möglich. Zur Behandlung dieses Problems wird die *Additive Analyse* eingeführt. Additive Analyse bedeutet, dass auf jeder Ebene der Entwicklung nur Teilaspekte analysiert werden, die in dieser Ebene relevant sind. Dabei werden Annahmen für Aspekte tieferer Ebenen getroffen, welche dann als Beweisverpflichtungen weiter gegeben werden. Es bedeutet aber auch, dass bereits geführte Beweise nicht noch einmal geführt werden müssen. Die Anwendung dieses Konzepts auf Verifikationstechniken stellt Abbildung 4.7(a) schematisch dar. In Abbildung 4.7(b) wird die Anwendung am Beispiel des Vorgehens in SuReal und damit der additiven Verifikation von Realzeiteigenschaften gezeigt.

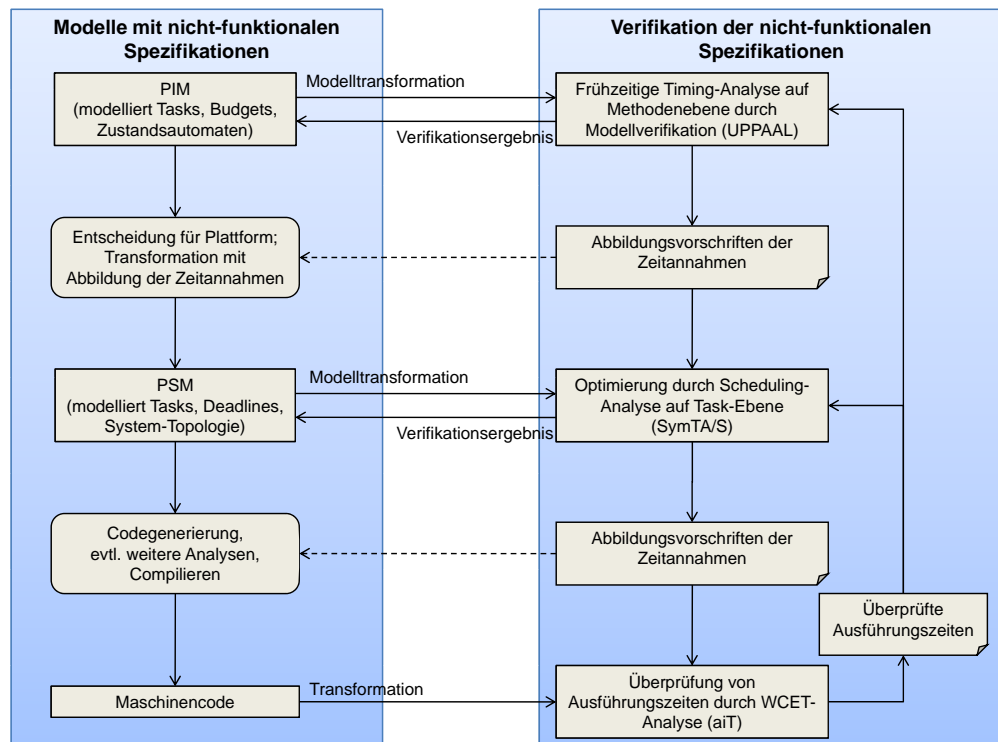
Folgende NFE-Prozessmuster wurden in diesem Bereich definiert:

- Verifikationsschleife
- Additive Verifikation der NFE

Neben der in dieser Arbeit fokussierten formalen Verifikation nicht-funktionaler Eigenschaften, können natürlich auch Tests und Simulationen zur Überprüfung eingesetzt werden. Nach einer entsprechenden Untersuchung der dazu notwendigen Prozessaktivitäten und Artefakte ist es möglich, dafür ebenfalls NFE-Prozessmuster zu definieren.



(a) Additive Verifikation als Anwendung des Konzepts der Additiven Analyse



(b) Beispiel für die Anwendung der Additiven Verifikation

Abbildung 4.7: Additive Verifikation der NFE

#### Abbildungsmechanismen

Um die verschiedenen Abstraktionsstufen der Softwareentwicklung zu verbinden, müssen Abbildungsmechanismen zwischen den einzelnen Verfeinerungs- bzw. Abstraktionsebenen von nicht-funktionalen Eigenschaften eingesetzt werden. Diese Abbildungen sind nicht nur auf der Modellierungsseite sondern auch auf der Seite der Überprüfung der NFE durch Verifikation, Simulation oder Testen notwendig. Daraus ergeben sich zwei Arten von Abbildungen bzw. Transformationen:

- *Vertikale Transformationen* beschreiben die Abbildung der NFE zwischen verschiedenen Abstraktionsstufen der Modellierung. Sie verfeinern das Systemmodell, d.h. der Informationsgehalt der Modelle wird erweitert. Mit der vertikalen Transformation der NFE sollte eine Analyse der Operationalisierung der Eigenschaften einhergehen.
- *Horizontale Transformationen* bilden dagegen Modelle innerhalb einer Abstraktionsebene ab. Im diskutierten Fall werden Modelle der Anwendungsmodellierung (z.B. UML State Charts) in prüfbare Modelle (z.B. Timed Automata) überführt. Der Detaillierungsgrad der Informationen bleibt dabei erhalten.

Damit wird erreicht, dass auch bei den Überprüfungsverfahren verschiedene Abstraktionsebenen bei der Entwicklung genutzt werden können und eine modellgetriebene Verfeinerung zwischen den Prüfmodellen möglich wird. Dies unterstützt außerdem die Additive Verifikation der nicht-funktionalen Eigenschaften eines Systems, da es die Weitergabe von Beweisverpflichtungen ermöglicht.

Folgendes NFE-Prozessmuster wurde daher in diesem Bereich definiert:

- Transformation der NFE-Spezifikationen

Weiterhin erleichtert die Verknüpfung der verschiedenen Abstraktionsebenen durch explizite Abbildungsbeschreibungen die Umsetzung der geforderten Rückverfolgbarkeit der NFE. Die Integration der Rückverfolgbarkeit erfordert ebenfalls eine Erweiterung des Vorgehensmodells, die in dieser Arbeit nicht näher betrachtet wird. Zur Definition eines entsprechenden NFE-Prozessmusters kann auf den Arbeiten von [ARNRSG06] und [AvEI06] aufgebaut werden.

#### Kompositionstechniken

Nicht-funktionale Eigenschaften liegen zu Beginn der Systementwicklung meist als globale Anforderungen durch den Nutzer vor. Es handelt sich oft um implizite oder informelle Nutzeraussagen wie: „Das System muss sicher gegenüber Angriffen von außen sein.“ Solche Eigenschaften sind keine rein nicht-funktionalen Anforderungen, sondern werden zum Teil durch Funktionalität umgesetzt. Durch eine entsprechende Analyse der Operationalisierbarkeit sind sie durch funktionale bzw. nicht-funktionale Eigenschaften zu konkretisieren und innerhalb des Entwicklungsprozesses kontinuierlich auf Systemeigenschaften abzubilden. Dazu gehört auch die Dekomposition der globalen Eigenschaften auf lokale Eigenschaften der verschiedenen Teile der Systemarchitektur. Die Problematik der NFE-Dekomposition liegt dabei in den konkreten Maßen, die für die expliziten Beschreibungen notwendig sind, da zum Zeitpunkt der Zerlegung die konkreten Bedingungen zur

Laufzeit noch nicht bekannt sind. Bei deren Zerlegung in lokale Eigenschaften von Komponenten und Methoden sind daher Annahmen zu treffen. Die notwendigen Aufgaben und Produkte wurden in den folgenden zwei NFE-Prozessmustern zusammengefasst:

- Analyse der Operationalisierung
- Dekomposition der NFE-Anforderungen

Zur Überwachung der globalen Anforderungen müssen die Annahmen, die bei der Dekomposition für die lokalen Eigenschaften festgelegt wurden, überprüft werden. Vergleichbar zum Vorgehen in der komponentenbasierten Entwicklung müssen die tatsächlichen lokalen nicht-funktionalen Eigenschaften aller Systemteile gemeinsam die ursprünglichen globalen Anforderungen an das Gesamtsystem erfüllen. Diese Komposition ist Bestandteil des Integrationstests und stellt die gegenläufige Aktivität zur Dekomposition dar. Den Zusammenhang zwischen lokalen und globalen NFE kann man durch Verträge beschreiben. Während in der Dekomposition Verträge über die dekomponierten Eigenschaften spezifiziert werden, werden diese Verträge nun auf deren Erfüllung überprüft. Dabei sind nicht nur Verträge zwischen den einzelnen Systemteilen (Komponenten) zu schließen, sondern auch zwischen den Komponenten und den verschiedenen Systemnutzern bzw. -betreibern [ZR04]. Dazu gehört auch die Beschreibung von Verträgen mit der Laufzeitumgebung zur Bereitstellung von Ressourcen. Im Falle der direkten Ausführung auf dem Betriebssystem ohne dazwischen liegende „Middleware“ handelt es sich dabei um Rechenzeit und Speicherbedarf. Grundlage aller Verträge ist die explizite Beschreibung von Schnittstellen und den zugehörigen nicht-funktionalen Eigenschaften. Folgendes NFE-Prozessmuster wurde in diesem Zusammenhang definiert:

- Komposition der NFE-Anforderungen

##### 4.3.5 Einordnung in ein Vorgehensmodell

Die Anwendung der in diesen vier Bereichen definierten Konzepte bildet den Kern der notwendigen Vorgehensmodellerweiterungen zur systematischen Behandlung und Umsetzung nicht-funktionaler Laufzeiteigenschaften. Wie in Abbildung 4.8 dargestellt, beschreiben die NFE-Prozessmuster dabei nur abstrakte Methoden, welche durch konkrete Verfahren zu instanzieren sind. Sie bilden damit den Ausgangspunkt für die projektspezifische Anpassung der erweiterten Vorgehensmodelle. Diese wird durch die Methodik der kontextbasierten Anpassung von Vorgehensmodellen für laufzeitkritische Software unterstützt. Die Methodik ist im Kapitel 5 beschrieben.

Wie Abbildung 4.8 zeigt, sind die Konzepte durch entsprechende Werkzeuge zu unterstützen sowie durch die Definition einer Ablaufstruktur innerhalb des Vorgehensmodells sinnvoll miteinander zu verknüpfen. Die Ablaufstruktur definiert dabei sowohl Phasen, Reihenfolge und Ergebnisse von Schritten als auch die ausführenden Rollen. Zusätzlich zu den bekannten Rollen der Anwendungsentwicklung, wie Analysten, Softwarearchitekten und Tester wurde die Rolle *Verifikationsexperte* definiert. Er ist für die Aktivitäten innerhalb des Prozessmusters Verifikationsschleife verantwortlich. Im



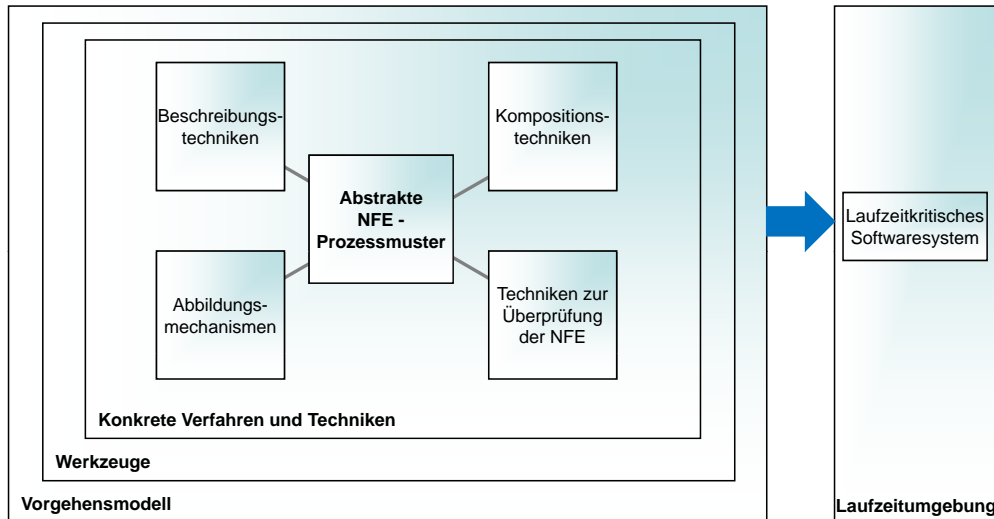


Abbildung 4.8: Bereiche der notwendigen Vorgehensmodellerweiterung

günstigsten Fall finden diese voll automatisiert und damit für den Entwickler transparent statt. Doch selbst dann sollte eine derartige Rolle zur Verfügung stehen, um bei fraglichen Analyseergebnissen beratend eingreifen zu können. Die Rollen sind im Detail in Anhang B beschrieben.

Die effizienten und risikoarmen Entwicklung laufzeitkritischer Systeme erfordert die inkrementelle und iterative Entwicklung des Softwaresystems entlang des kritischen Pfades. Der kritische Pfad beschreibt die Verkettung derjenigen Systemfunktionen, die untrennbar mit der Erfüllung der geforderten nicht-funktionalen Eigenschaften zusammenhängen. Im Falle von Zeitanforderungen ist es die Kette von Funktionsaufrufen, welche in der Summe die längste Dauer aufweist. Selbst wenn die Entwicklung mit einem kritischen Szenario begonnen wird, ist die Erfüllung nicht-funktionaler Laufeigenschaften eines Systems durch das Hinzufügen neuer Funktionalität nicht zwangsläufig sichergestellt. So benötigt beispielsweise jede zusätzliche Funktion Rechenzeit, was sich wiederum negativ auf zeitbasierte Eigenschaften anderer Systemfunktionen auswirken kann. Daher ist für die Entwicklung der laufzeitkritischen Komponenten bzw. Subsysteme ein inkrementelles Vorgehen einzusetzen. Diese Aspekte werden durch folgende NFE-Prozessmuster definiert:

- Iterative Entwicklung entlang des kritischen Pfades
- Inkrementelle Entwicklung der kritischen Systembereiche

Beide Muster sind nach Tabelle 4.1 als NFE-Prozessmuster für Phasen einzuordnen. Voraussetzung für das Festlegen des kritischen Pfades ist eine Analyse der kritischen Szenarien eines Systems. Dabei sind nicht alle möglichen Abläufe innerhalb eines Szenarios als kritisch einzustufen. Der kritische Pfad ist nur eine mögliche Sequenz eines Kontroll-

oder Datenflusses, welche in einem zusätzlichen Analyseschritt extrahiert werden muss. Dieser ist im NFE-Prozessmuster:

- Identifikation der kritischen Szenarien

als Aktivitätsmuster definiert.

Eine Garantie über die Einhaltung der nicht-funktionalen Anforderungen während der Laufzeit des entwickelten Softwaresystems kann nur gegeben werden, wenn eine Laufzeitumgebung existiert, welche in der Lage ist Betriebsmittel zu planen und entsprechend bereitzustellen bzw. zu reservieren (Abbildung 4.8). Die dafür notwendige Behandlung nicht-funktionaler Eigenschaften zur Laufzeit des Systems findet derzeit in existierenden Prozessmodellen und -methoden keine Berücksichtigung. Zur Durchsetzung dieser Anforderungen ist es beispielsweise notwendig Systemressourcen auszuhandeln, zu priorisieren und gegebenenfalls zu reservieren. Eine mögliche Umsetzung wurde im Projekt COMQUAD<sup>1</sup> entwickelt und ist ausführlich in [GPRZ04] und [HZP<sup>+</sup>07] beschrieben.

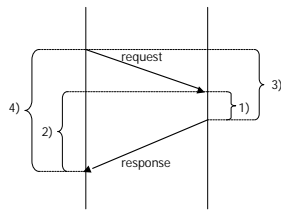
Die detaillierte Beschreibung der hier definierten NFE-Prozessmuster ist in Anhang A zu finden. Wie bei anderen Typen von Mustern (z.B. Entwurfsmustern), ist die Auflistung individuell erweiterbar. Beim Einsatz der in dieser Arbeit definierten Muster wird eine systematische und von den Anforderungen bis zur Ausführung durchgängige Behandlung der NFE ermöglicht. Das dadurch entstehende Vorgehensmodell wird als *erweitertes Vorgehensmodell zur Entwicklung laufzeitkritischer Software* bezeichnet. Voraussetzung dafür ist deren entsprechende Umsetzung in konkrete Verfahren, welche aufeinander abgestimmt sind. Deren Bereitstellung ist nicht Inhalt dieser Arbeit. Vielmehr wird durch die systematische Definition eines Vorgehensmodells für die Entwicklung von Software mit laufzeitkritischen nicht-funktionalen Eigenschaften aufgezeigt, welche Aktivitäten, Methoden sowie Techniken zur Verfügung zu stellen sind, um das Ziel der durchgängigen Behandlung dieser Eigenschaften erreichen zu können. Dies wiederum bildet eine strukturierte Ausgangsbasis für das Füllen der Lücken durch weitere Forschungs- und Entwicklungsarbeiten.

#### 4.3.6 NFE-Prozessmuster in der NFE-Domänenentwicklung

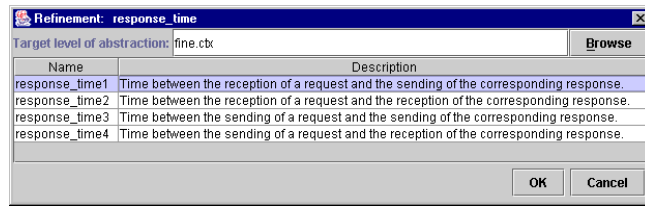
Bei den bisher vorgestellten NFE-Prozessmuster handelt es sich um Muster die in der Anwendungsentwicklung eingesetzt werden. Für deren Anwendung müssen vorab verschiedene Sprachmittel bereitgestellt werden. Dazu gehören sowohl Modellierungssprachen (z.B. ein UML Profil für Realzeitspezifikationen) sowie Transformationssprachen zur Beschreibung der Abbildungen der NFE auf weniger abstrakte Ebenen (z.B. die Abbildung aus dem UML-Modell auf Real Time Java Code) bzw. Analysemodelle (z.B. die Abbildung vom UML Modell in einen verifizierbares Automatenmodell). Hier kann in Anlehnung an die MDA in Domänen- und Anwendungsentwicklung ([GPR06], Domain- und Application Engineering) unterschieden werden. Da es sich bei der betrachteten Domäne um die Umsetzung nicht-funktionaler Anforderungen handelt, wird von der NFE-Domäne gesprochen. Zusätzlich zu den bereits angeführten NFE-Prozessmustern

---

<sup>1</sup>COMponents with QUantitative properties and ADaptivity. DFG-geförderte Forschergruppe, TU Dresden, LMU München, TU Erlangen, 2001-2004.



(a) Verschiedene Arten



(b) Beispiel eines Auswahldialogs

Abbildung 4.9: 1:n-Abbildung von NFE am Beispiel einer Antwortzeitspezifikation

der Anwendungsentwicklung werden folgende Muster der NFE-Domänenentwicklung als Prozessmuster für Aktivitäten definiert:

- NFE-spezifische Sprache
- NFE-spezifische Abbildungsbeschreibungen

Beide NFE-Prozessmuster sind ebenfalls in Anhang A beschrieben.

Ein wesentlicher Bestandteil des Musters „NFE-spezifische Abbildungsbeschreibungen“ ist die Bereitstellung einer entsprechenden Werkzeugunterstützung für die Anwendung der Abbildungsvorschriften innerhalb von Transformationsschritten in der Anwendungsentwicklung. In die Definition der Abbildungsmechanismen der NFE müssen sowohl Aspekte der strukturellen Verfeinerung als auch der Verfeinerung der NFE und damit der zugehörigen Maßspezifikation eingehen. Diese Art der Verfeinerung wird als *Maßverfeinerung* (Measurement Refinement, [RZ04a]) bezeichnet. Bei dieser Art der Verfeinerung muss die Möglichkeit einer 1:n-Abbildung der zugrundeliegenden Maßspezifikation einer nicht-funktionalen Eigenschaft beachtet werden. Abbildung 4.9 zeigt dies am Beispiel einer Antwortzeitspezifikation. Während es auf einer abstrakteren Ebene der Systemmodellierung ausreicht, diese als Zeit zwischen dem Start und dem Ende eines Methodenaufrufs zu interpretieren, muss dies auf einer konkreteren Modellierungsebene präzisiert werden. Dabei können in diesem Beispiel vier Möglichkeiten unterschieden werden (Abbildung 4.9(a)): Eine Antwortzeit ist die Zeit zwischen

- 1) Empfang einer Anfrage (request) und dem Senden der zugehörigen Antwort (response)
- 2) Empfang einer Anfrage und dem Empfang der zugehörigen Antwort
- 3) Senden einer Anfrage und dem Senden der zugehörigen Antwort
- 4) Senden einer Anfrage und dem Empfang der zugehörigen Antwort

Diese Möglichkeiten sind in den Abbildungsbeschreibungen entsprechend zu verankern. Für den Nutzer der Transformation innerhalb der Anwendungsentwicklung sind

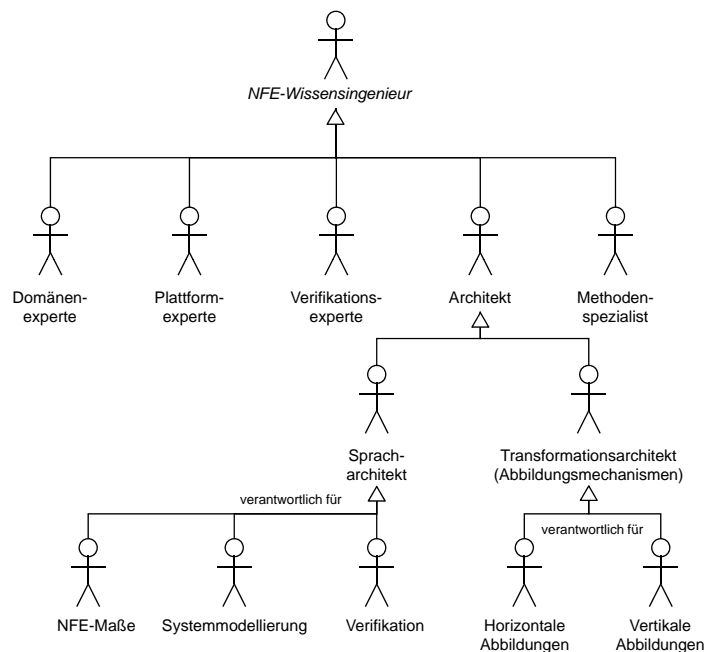


Abbildung 4.10: Übersicht über die Spezialisierungen der Rolle „NFE-Wissensingenieur“

die zugehörigen Maßdefinitionen transparent zu halten. Ihm sind lediglich eindeutige und nachvollziehbare Auswahlkriterien innerhalb der Werkzeugumgebung anzubieten. Ein Beispiel für einen entsprechenden Auswahldialog zeigt Abbildung 4.9(b). In [RZ07] wird diese Problematik ausführlicher diskutiert.

Die Bereitstellung der geforderten Sprachen und damit der Einsatz der beiden NFE-Prozessmuster der Domänenentwicklung liegt in der Verantwortung der Rolle NFE-Wissensingenieur. Diese Rolle lässt sich in verschiedene Spezialisierungen aufteilen. Abbildung 4.10 gibt einen Überblick über die Rollen. So gibt es Experten für Domäne, Plattform und Verifikation, welche dafür verantwortlich sind, die jeweiligen Konzepte zu abstrahieren und als Modell bereitzustellen. Die Rolle des Spracharchitekten entwickelt auf Basis dieser Modelle Beschreibungssprachen. Diese Rolle besitzt ebenfalls spezifische Ausprägungen, deren Namen entsprechend ihre Verantwortlichkeit symbolisieren. Eine detaillierte Beschreibung der einzelnen Rollen befindet sich im Anhang B.

### 4.4 Zusammenfassung

Einführend wird der Stand der Technik auf dem Gebiet der Vorgehensunterstützung für laufzeitkritische Softwareentwicklungen vorgestellt. Die Analyse zeigte Defizite und Lücken in der kontinuierlichen Prozessunterstützung nicht-funktionaler Systemanforderungen auf. Darauf aufbauend wurden notwendige Eigenschaften eines erweiterten Vorgehensmodells aufgestellt und im Anschluss NFE-Prozessmuster definiert. Durch die

Erweiterung des Softwareentwicklungsprozesses um NFE-Prozessmuster werden in das ursprüngliche Vorgehensmodell Methoden eingeführt, welche dazu dienen, die Umsetzung nicht-funktionaler Laufzeiteigenschaften systematisch zu unterstützen.

Die Integration der NFE-Prozessmuster in Vorgehensmodelle und deren Anpassung an konkrete Projektbedingungen unterliegt einer eigenen Methodik, welche im folgenden Kapitel eingeführt wird. Der Nachweis, dass die in dieser Arbeit definierten NFE-Prozessmuster die geforderten Eigenschaften eines erweiterten Vorgehensmodells zur systematischen Entwicklung von Software mit nicht-funktionalen laufzeitkritischen Eigenschaften erfüllt, wird im Evaluierungskapitel (Kapitel 9) geführt.



## 5 PROKRIS-Methodik: Kontextbasierte Anpassung der erweiterten Vorgehensmodelle für laufzeitkritische Software

Die vorliegenden NFE-Prozessmuster definieren ausschließlich abstrakte Methoden zur Unterstützung nicht-funktionaler Laufzeiteigenschaften auf der Makroprozessebene. Die durch diese Muster erweiterten Vorgehensmodelle sind, um sie in konkreten Entwicklungsprojekten einsetzen zu können, an die projektspezifischen Gegebenheiten anzupassen. Zur Unterstützung dieser Anpassung der Makroprozesse auf die Mikroprozessebene wurde die Methodik der *Kontextbasierten Anpassung der erweiterten Vorgehensmodelle für die Entwicklung laufzeitkritischer Software* entwickelt. Da die Methodik durch das *PROKRIS-Framework* unterstützt wird, wird sie kurz als *PROKRIS-Methodik* bezeichnet. Der Einsatz der Methodik ist durch die Modellierung von Vorgehensmodellen unterschiedlicher Abstraktionsgrade gekennzeichnet. Kernkonzepte sind die Wiederverwendung von Teilprozessen, die Verbindung der Vorgehensmodelle der verschiedenen Ebenen durch Verfeinerungs- und Anpassungsschritte sowie eine Ausführungsunterstützung zur Laufzeit innerhalb eines konkreten Projekts. In diesem Kapitel wird die Methodik im Detail vorgestellt.

Das Kapitel beschreibt die Methodik aus Sicht der Anwendung. Dabei werden als Beispiel überwiegend Auszüge aus dem Vorgehensmodell des SuReal-Projekts eingesetzt und die Werkzeugumgebung des PROKRIS-Frameworks genutzt. Die vollständige Modellierung der einzelnen Abstraktionsstufen dieses Modells kann Anhang D entnommen werden.

Die Erstellung der erweiterten NFE-Vorgehensmodelle mit der PROKRIS-Methodik beruht hauptsächlich auf prozessrelevanten Entscheidungen, wird aber außerdem von Architekturentscheidungen beeinflusst. Durch die Aufteilung einer Softwarearchitektur in verschiedene Subsysteme entstehen innerhalb des Vorgehensmodells Subprozesse. Dieser Aspekt wird kurz diskutiert.

### 5.1 Konzepte der PROKRIS-Methodik

Die PROKRIS-Methodik hat zum Einen das Ziel, die projektspezifische Anpassung der, durch die NFE-Prozessmuster erweiterten, abstrakten Vorgehensmodelle zu unterstützen. Ein zweites Ziel ist die konkrete Unterstützung bei der Anwendung des angepassten Vorgehensmodells zur Projektlaufzeit. Dazu stellt die Methodik verschie-

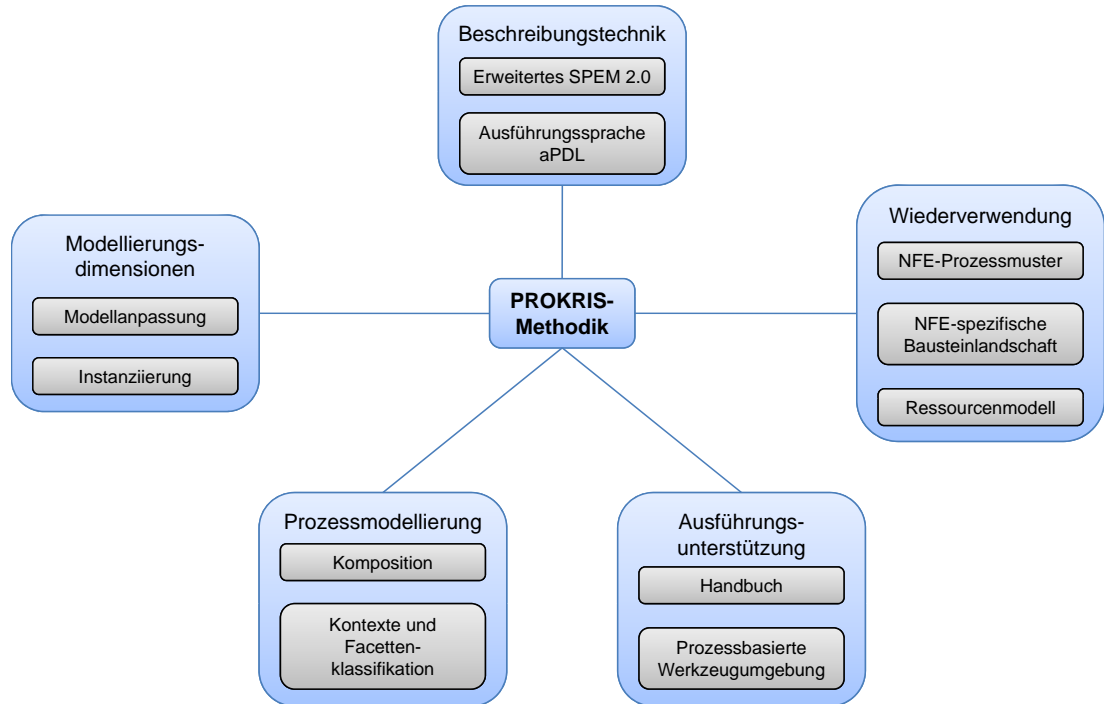


Abbildung 5.1: Die grundlegenden Konzepte der Methodik

denste Konzepte zur Verfügung, welche schematisch in Abbildung 5.1 dargestellt sind. Das Basiskonzept ist die Definition zweier *Modellierungsdimensionen* von Prozessspezifikationen, der Modellanpassung und der Instanziierung, um den beiden oben genannten Zielen zu entsprechen. Ein weiterer Aspekt ist die *Wiederverwendung* von Prozessspezifikationen. Dabei können sowohl die NFE-Prozessmuster wiederverwendet werden, als auch deren Instanzen, d.h. die durch konkrete Techniken hinterlegten und damit an spezifische NFE angepassten Prozessmusterinstanzen. Außerdem kann die Spezifikation der organisationspezifischen Ressourcen (Menschen, Werkzeuge und Daten) in verschiedenen Prozessmodellen eingesetzt werden. Ein drittes Konzept ist die *Prozessmodellierung*. Dieses besteht aus der Komposition von Vorgehensmodellen aus NFE-Prozessmustern bzw. -bausteinen, welche über einen Filteralgorithmus basierend auf Kontextinformationen arbeitet. Die *Ausführungsunterstützung* wiederum setzt sich aus der Bereitstellung von Handbüchern des Prozesswissens und von prozessbasierten Werkzeugumgebungen zusammen. Der Einsatz der vorab erwähnten Konzepte ist nur durch die Bereitstellung einheitlicher Sprachkonzepte möglich (*Beschreibungstechnik*). Die Methodik arbeitet mit einer erweiterten Version des Prozessmetamodells SPEM 2.0 und der ausführbaren Workflowsprache aPDL, einer Erweiterung von jPDL.

In den folgenden Abschnitten werden die einzelnen Konzepte erläutert.



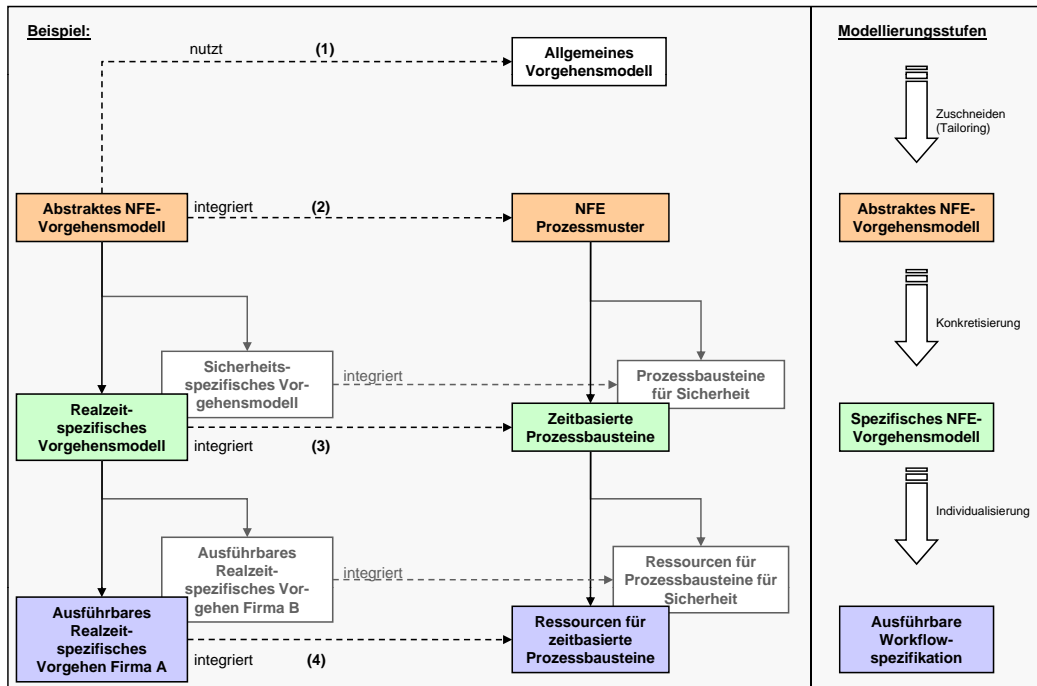


Abbildung 5.2: Abstraktionsebenen an einem Beispiel

## 5.2 Modellierungsdimensionen

Innerhalb des PROKRIS-Frameworks werden zwei verschiedene Dimensionen der Modellierung von Prozessspezifikationen unterschieden: die Modellanpassung und die Instanziierung.

### 5.2.1 Modellanpassung

Die Anpassung eines Vorgehensmodells an die spezifischen Randbedingungen eines Projekts stellt eine Dimension der Modellierung innerhalb der PROKRIS-Methodik dar. Im Falle des PROKRIS-Frameworks wird das Modell so angepasst, dass es auf die Umsetzung von nicht-funktionalen Anforderungen spezialisiert wird.

Im Folgenden soll kurz das Ziel dieser kontextbasierten Anpassung demonstriert werden. Abbildung 5.2 stellt den Ablauf schematisch dar. Als Beispiel dient das bereits eingeführte NFE-Prozessmuster Verifikationsschleife und dessen Umsetzung bezüglich Zeit- und Sicherheitseigenschaften. Das Muster beschreibt, wie Verifikationstechniken im Allgemeinen als Teilprozess in einen Entwicklungsprozess zu integrieren sind, um Eigenschaften eines modellierten Systems zu verifizieren. Das Muster an sich definiert dabei die notwendigen Aktivitäten und Arbeitsprodukte auf der abstrakten Ebene der Makroprozesse. Je nach dem, ob es sich beispielsweise um Sicherheits- oder Realzeitanforderungen handelt, sehen die konkreten Artefakte anders aus. So kann das Muster

Performanz	Zeit	CPU-Bedarf
		Antwortzeit
		Reaktionszeit
		Verzögerungszeit
	Leistung	Bandbreite
		Durchsatz
		Last
	Speicher	Puffergröße
		Speicherbedarf

Tabelle 5.1: Klassifikation von Performanzeigenschaften

Verifikationsschleife für Zeit an die in [RZ07] und für Sicherheit an die in [Pop05] beschriebenen Techniken angepasst werden (Abbildung 5.2, zeitbasierte Prozessbausteine bzw. Prozessbausteine für Sicherheit).

Die Verschiedenheit der NFE ist im Beispiel zu Anschauungszwecken stark ausgeprägt, so dass auf den ersten Blick argumentiert werden kann, dass diese Unterscheidung schon zu Projektstart vorhersehbar ist. Die Stärken dieser Trennung werden deutlicher, wenn man allein die Verschiedenartigkeit von Realzeiteigenschaften betrachtet. Eine Beispiel für eine Realzeiteigenschaft ist Performanz. Diese lässt sich in einer ersten Unterteilung in Zeit-, Last- und Speichereigenschaften unterteilen [Ban04]. Die weitere Klassifikation ist in Tabelle 5.1 dargestellt. Es handelt sich dabei um verschiedenste nicht-funktionale Eigenschaften, deren Verifikation sehr differenziert umzusetzen ist.

Auf Basis der allgemeinen NFE-Prozessmuster wird das abstrakte NFE-Vorgehensmodell erstellt. Im Beispiel wird das NFE-Prozessmuster Verifikationsschleife neben weiteren Prozessmustern als Erweiterung integriert (Abbildung 5.2, (2)). Es ist außerdem möglich, das abstrakte Modell durch entsprechendes *Tailoring* in ein bereits existierendes Vorgehensmodell einzubinden (1).

Nach einer genaueren Analyse der Anforderungen ist es möglich, die NFE-Prozessmuster und damit die Methoden durch konkrete Verfahren und Techniken zu konkretisieren. Die *Konkretisierung*<sup>1</sup> unterstützt dies durch die Bereitstellung NFE-spezifischer Prozessbausteine innerhalb einer Prozessbibliothek. Diese spezifischen Bausteine sind Instanziierungen der abstrakten NFE-Prozessmuster. Sie ersetzen die generischen Methoden durch konkrete Verfahren für die jeweilige nicht-funktionale Eigenschaft. Konkretisiert der Prozessingenieur beispielsweise das NFE-Prozessmuster Verifikationsschleife im Kontext von Realzeitbedingungen, erhält er ein *Realzeitspezifisches Vorgehensmodell* (Abbildung 5.2, (3)).

Doch auch nach dieser Spezialisierung auf konkrete Qualitätseigenschaften ist der Prozess ein statisches Modell. Im Kontext kritischer Systemeigenschaften ist es sinnvoll, Teilprozesse eines Vorgehensmodells innerhalb eines Projekts ausführbar zu machen. Erst dadurch ist eine automatisierte Unterstützung der Entwickler bei der Umsetzung

<sup>1</sup> Als Konkretisierung wird die Anwendung eines abstrakten Konzeptes auf eine spezifische Situation bezeichnet.

des Prozesses durch eine Ausführungsumgebung möglich. Neben dem Bereitstellen einer Ausführungssemantik ist dafür die zusätzliche Spezifikation der Organisationsstruktur innerhalb des Prozessmodells notwendig. Dies geschieht durch den Schritt der *Individualisierung*, in dem Rollen durch konkrete Personen oder Personengruppen und allgemeine Werkzeugbeschreibungen durch konkrete Werkzeuge (z.B. UML Werkzeug durch konkretes UML-Werkzeug „TopCased“) ersetzt werden. Dadurch können die Prozessmodelle an die Gegebenheiten verschiedener Firmen, Projekte und Entwicklerteams angepasst werden. Abbildung 5.2 zeigt diesen letzten Schritt (4) in dem die konkreten Ressourcen einer konkreten Firma in den Prozess integriert werden.

Auf der rechten Seite ordnet Abbildung 5.2 das Beispiel in die drei Stufen *Abstraktes NFE-Vorgehensmodell* als Makroprozessmodell, *Spezifisches NFE-Vorgehensmodell* als Mikroprozessmodell und die *Ausführbare Workflowspezifikation* als Ausführungsmodell ein. Die Modellanpassung verbindet diese drei Abstraktionsebenen der Prozessmodellierung durch die drei Verfeinerungsschritte: *Tailoring*, *Konkretisierung* und *Individualisierung*. Die Unterscheidung der drei Stufen lässt sich folgendermaßen zusammenfassen:

*Tailoring*: Komposition von bestehenden Vorgehensmodellen und NFE-Prozessmustern zum abstrakten NFE-Vorgehensmodell

*Konkretisierung*: zu NFE-spezifischen Vorgehensmodellen durch Ersetzen der NFE-Prozessmuster mit NFE-spezifischen Instanzen, den NFE-Prozessbausteinen

*Individualisierung*: durch Binden konkreter Ressourcen (Personen, Daten, Werkzeuge) an die abstrakten Ressourcenschnittstellen der Aufgaben innerhalb des Vorgehensmodells.

Bei der detaillierten Betrachtung der Modellanpassung ist es außerdem wichtig, zwischen *statischen* und *ausführbaren Vorgehensmodellen* zu unterscheiden:

**Statische Vorgehensmodelle** sind Vorgehensmodelle, welche durch umfassende Beschreibung der Aufgaben, Artefakte und Rollen sowie Methoden und Techniken spezifiziert sind. Dazu gehört eine Beschreibung der Arbeitsabläufe und des Datenflusses durch die Zuordnung von eingehenden und ausgehenden Artefakten einer Aufgabe. Sie sind als *präskriptive*<sup>2</sup> Modelle einzustufen, das bedeutet, sie beschreiben, was das Vorgehensmodell beinhalten soll [Sca02]. Dies wird durch die Bereitstellung des Handbuchs untermauert. Im PROKRIS-Framework sind die abstrakten und die NFE-spezifischen Vorgehensmodelle als statisch einzustufen.

**Ausführbare Vorgehensmodelle** sind Vorgehensmodelle, welche den genauen Ablauf eines Entwicklungsprozesses beschreiben. Sie sind als *deskriptive Modelle* einzustufen, da sie beschreiben, wie das Vorgehensmodell tatsächlich eingesetzt wird [Sca02]. Im PROKRIS-Ansatz ist hier die ausführbare Workflowspezifikation einzustufen.

<sup>2</sup>In Anlehnung an präskriptive und deskriptive Grammatiken wird in der englischen Literatur zu Softwareprozessmodellen in *prescriptive* und *descriptive process models* unterschieden.

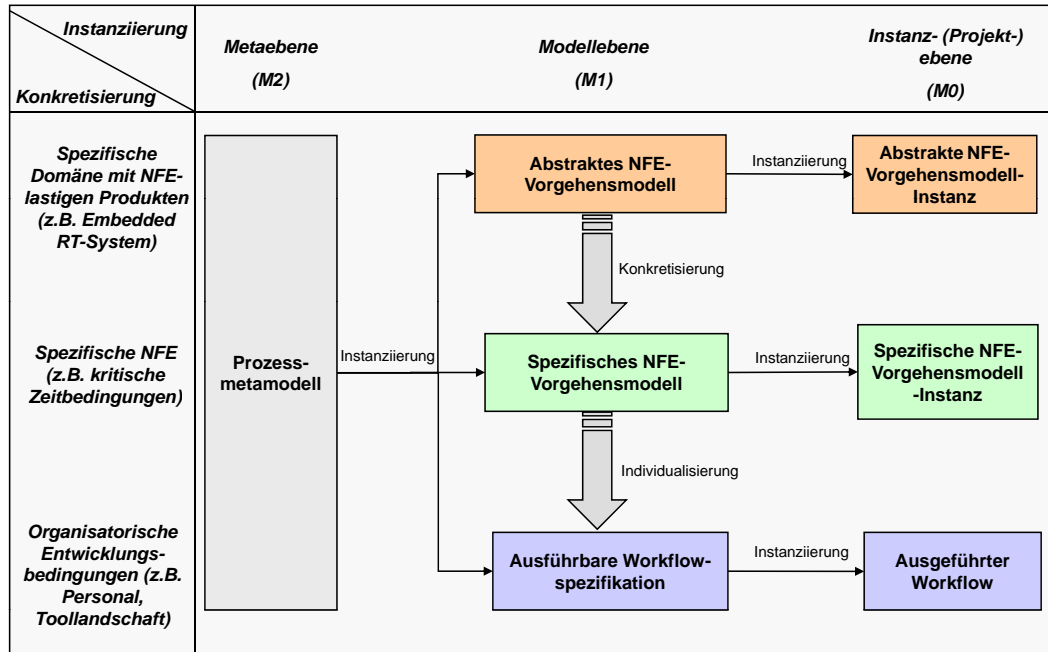


Abbildung 5.3: Konkretisierungs- und Modellierungsebenen

Anzumerken sei noch, dass im Falle der hier vorgestellten Methodik die Begriffe Verfeinerung und Modellanpassung als Synonyme gelten. Dies ist möglich, da es sich bei der Anpassung um eine Anreicherung der Vorgehensmodelle mit spezifischerem Wissen im Sinne einer Modellverfeinerung handelt. Dies ist nicht zu verwechseln mit der Verfeinerung im mathematischen Sinn.

### 5.2.2 Instanziierung

Orthogonal zur Modellanpassung von Vorgehensbausteinen bzw. gesamten Vorgehensmodellen existiert die Modellierungsdimension der Instanziierung von Vorgehensmodellen. Wie in der objektorientierten Modellierung üblich, können auch Prozessmodelle auf verschiedenen Ebenen der Instanziierung angesiedelt sein. In Anlehnung an [RFKR00] werden in Abbildung 5.3 die verschiedenen Abstraktionsebenen im Kontext des PROKRIS-Frameworks eingeordnet. Es können drei Abstraktionsstufen unterschieden werden: die Metaebene, die Modellebene und die Instanzebene. Dabei handelt es sich um die schon in Kapitel 2, Abbildung 2.5 beschriebenen Metaebenen der Modellierung. Auf der Metaebene befindet sich das Prozessmetamodell. Dieses stellt die Sprache bereit, mit der auf der nächst tieferen Ebene, der Modellebene, ein Prozessmodell beschrieben werden kann. Durch Instanziierung der im Prozessmodell spezifizierten Elemente mit konkreten Objekten aus der realen Welt erhält man eine projektspezifische Beschreibung eines Prozessmodells. Man befindet sich nun auf der Instanz- bzw. Projektebene der Modellierung.

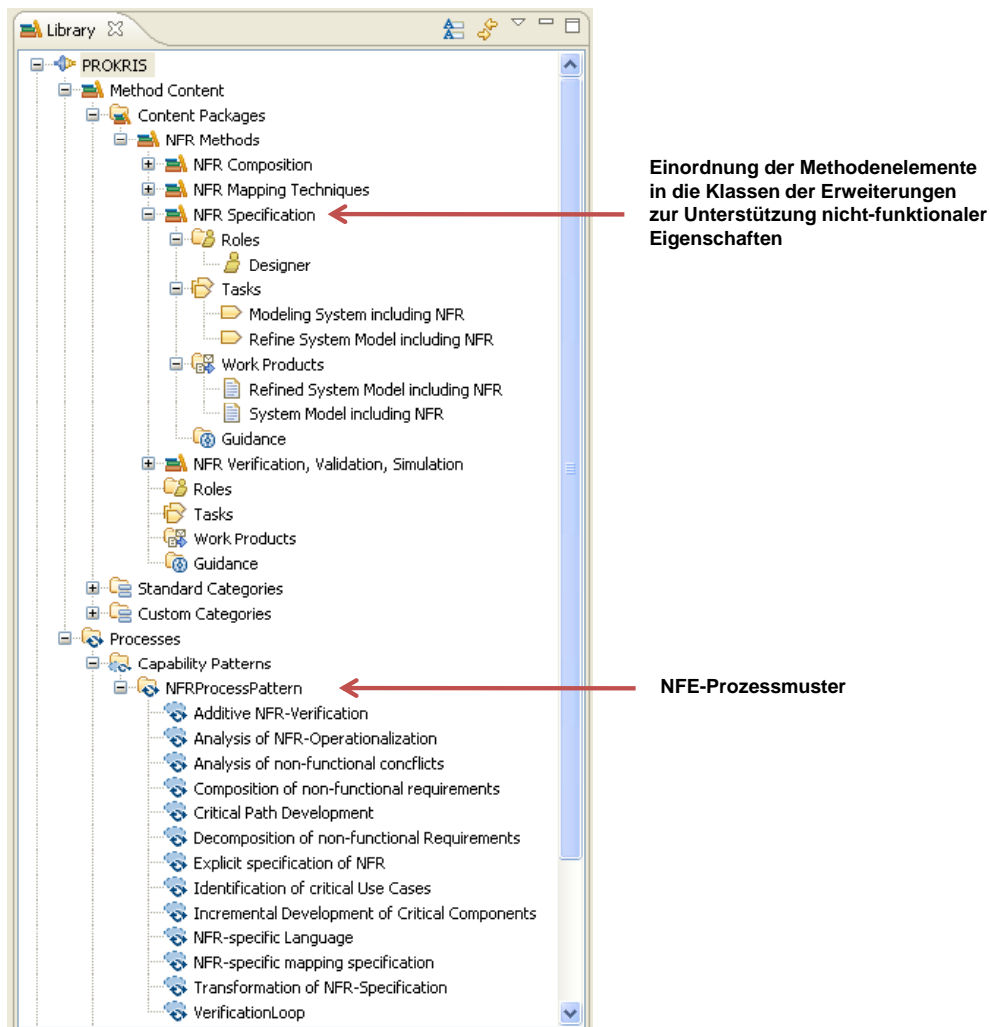


Abbildung 5.4: Auszug aus der Struktur der PROKRIS-Prozessbibliothek

### 5.3 Einheitliche Beschreibungstechnik

Um Wiederverwendung und Komposition von NFE-Prozessmustern und -bausteinen zu ermöglichen, müssen diese in einer einheitlichen Beschreibungssprache auf Basis eines Metamodells vorliegen.

Die Unterscheidung in statische und ausführbare Vorgehensmodelle spiegelt sich in den verwendeten Sprachen wieder. SPEM 2.0 stellt eine geeignete Grundlage für die Modellierung der statischen Modelle dar, da der Fokus auf einer allumfassenden Beschreibung des Vorgehensmodells liegt. Defizite weist es vor allem in der Unterstützung der Konkretisierung von Prozessmodellen, das heißt dem Beschreiben von Abbildungen zwischen Prozessmustern und deren Instanzen, sowie der Abbildung auf eine Ausführungsspezifikation auf. Daher wird für das PROKRIS-Framework eine entsprechende Erweiterung definiert.

Als Ausführungssprache setzt das PROKRIS-Framework aPDL, eine Erweiterung von jPDL [jPD] ein. jPDL ist die Ausführungssprache der Workflowmaschine jBPM [jBP], deren Grundzüge in Kapitel 2.8 erläutert wurden. Die Erweiterungen umfassen im Wesentlichen die Integration von Ressourcenkonzepten und Werkzeuginitialisierung.

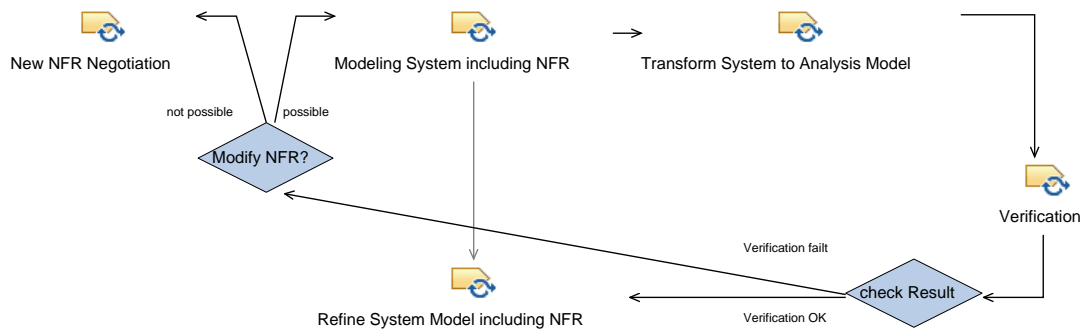
### 5.4 Wiederverwendung

Einen weiteren Aspekt des PROKRIS-Frameworks stellt die Wiederverwendung dar. Dies beinhaltet sowohl die Wiederverwendung der NFE-Prozessmuster und der durch die Konkretisierung entstehenden NFE-Prozessbausteinlandschaft als auch der Beschreibung der Organisationsstruktur in Form von Ressourcenspezifikationen.

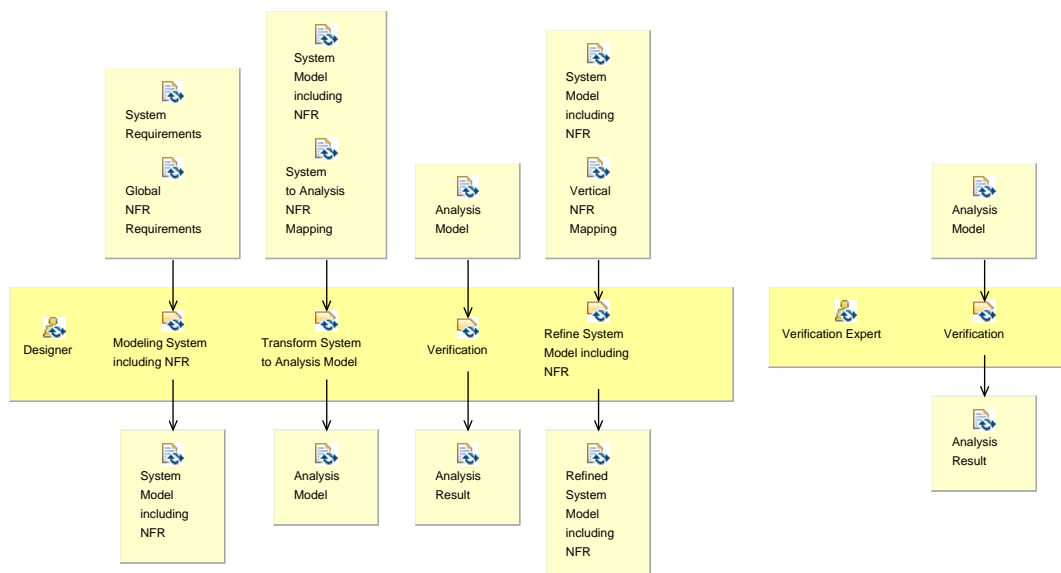
#### 5.4.1 NFE-Prozessmuster

Die in Kapitel 4 beschriebenen NFE-Prozessmuster bilden den Kern der PROKRIS-Methodik. Sie können zur Komposition eines abstrakten NFE-Vorgehensmodells verwendet werden. Die NFE-Prozessmuster werden in der PROKRIS-Prozessbibliothek beschrieben. Abbildung 5.4 zeigt einen Auszug aus der Struktur im Editor der PROKRIS-Bibliothek. Die NFE-Prozessmuster sind im unteren Teil des Baums, mit Hilfe des neu definierten UML-Profil-Elements `NFProcessPattern`, eingeordnet. Die Modellierung erfolgt unter Nutzung der Elemente des Methodeninhalts, wie Aufgaben, Rollen und Arbeitsprodukte. Diese sind ebenfalls in der Prozessbibliothek strukturiert abgelegt (Abbildung 5.4, oben). Die Struktur entspricht den in Kapitel 4 (Abbildung 4.8) identifizierten Klassen von Vorgehenserweiterungen für Software mit lauffzeitkritischen Eigenschaften: Beschreibungstechniken (`NFR Specification`), Kompositionstechniken (`NFR Composition`), Abbildungsmechanismen (`NFR Mapping Techniques`) und Analysetechniken (`NFR Verification, Validation and Simulation`). Die Abbildung zeigt die Einordnung beispielhaft für ausgewählte Elemente der Beschreibungstechniken.

Das Prozessmuster beschreibt nicht nur die Elemente sondern auch deren Beziehungen. Abbildung 5.5 zeigt als Beispiel das Modell des Musters Verifikationsschleife. Unter Nutzung des erweiterten SPEM 2.0 Profils ist das Aktivitätsdiagramm sowie die Zuordnung von Artefakten und Rollen zu den einzelnen Aufgaben des Musters dargestellt.



(a) Aktivitätsdiagramm



(b) Zuordnung Rollen, Aktivitäten und Arbeitsprodukte

Abbildung 5.5: NFE-Prozessmuster Verifikationsschleife

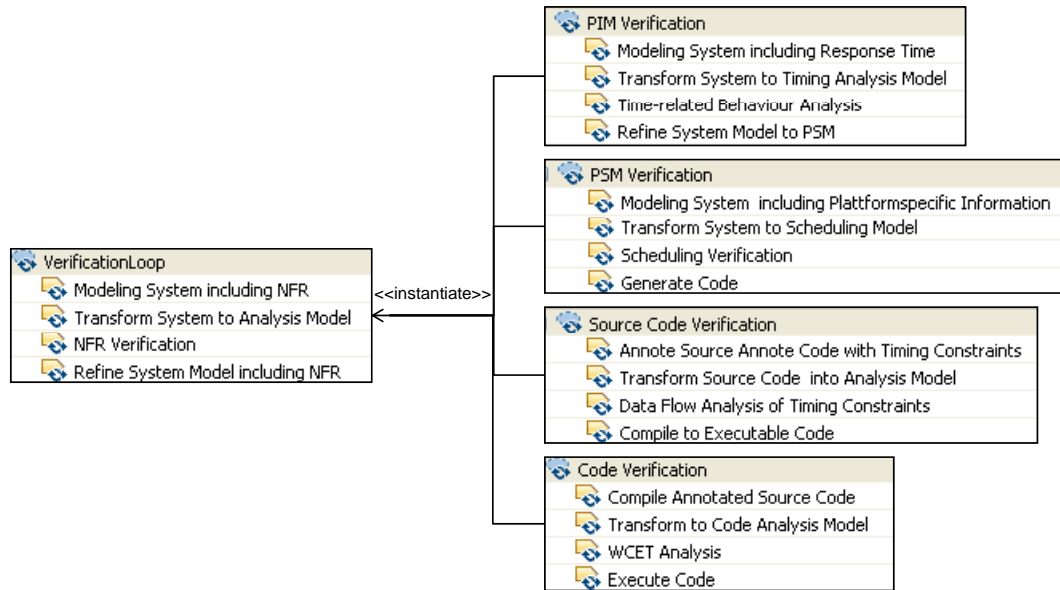


Abbildung 5.6: Instanzen des Musters Verifikationsschleife für Realzeitaspekte

So ist beispielsweise die Aufgabe `Task::Modeling System including NFR` aus Abbildung 5.4 sowohl im Aktivitätsdiagramm als auch in der Zuordnung der Rollen und Artefakte enthalten. Ebenfalls zu sehen ist die Zuordnung der Rolle `Role::Designer` und des Artefakts `Work Products::Refined System Model including NFR`.

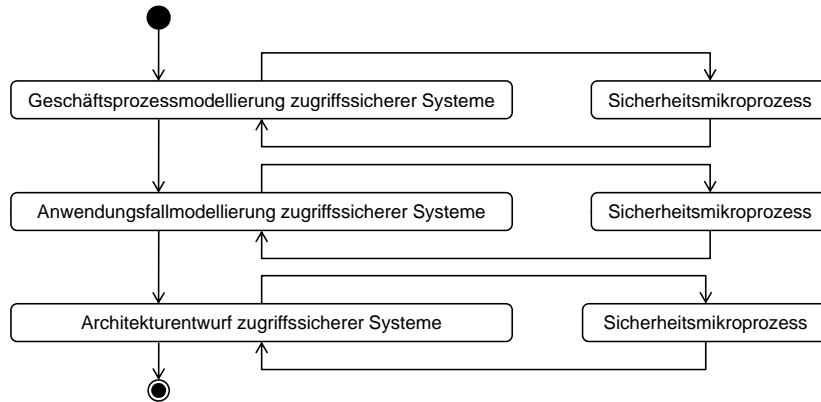
#### 5.4.2 NFE-spezifische Bausteine

NFE-spezifische Prozessbausteine modellieren an verschiedene Randbedingungen angepasste Instanzen der NFE-Prozessmuster. Die wichtigste Randbedingung ist die durch das System zu gewährleistende laufzeitkritische Eigenschaft. Aber auch die Abstraktionsstufe der Modellierung, wie plattformunabhängige oder plattformabhängige Modellierung hat einen Einfluss auf die Umsetzung des Prozessmusters. Im Folgenden sollen beispielhaft die Instanzen des NFE-Prozessmusters Verifikationsschleife für Realzeit bzw. Sicherheit diskutiert werden, welches in Abbildung 5.6 auf der linken Seite als Ablaufstruktur der Aktivitäten des Musters dargestellt ist.

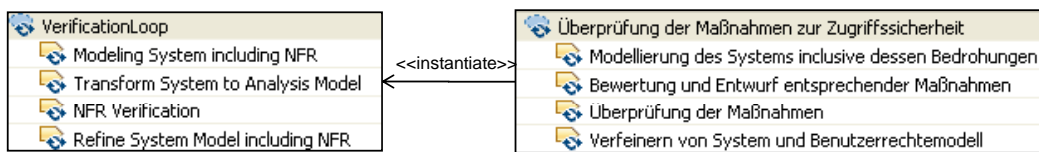
##### Realzeit-spezifische Bausteine am Beispiel von Antwortzeit

Abbildung 5.6 zeigt die realzeitspezifischen Instanzen des NFE-Prozessmusters Verifikationsschleife, bei deren Anwendung speziell die Einhaltung von hart definierten Antwortzeiten überprüft werden kann. Die Instanzen des NFE-Prozessmusters sind auf die Abstraktionsebenen der Systemerstellung und damit die Entwicklungsphasen spezialisiert. Es können vier phasenspezifische Umsetzungen des Musters unterschieden werden: plattformunabhängige Modellierung (**PIM Verification**), plattformabhängige Modellierung (**PSM Verification**), Implementierung bzw. Codegenerierung (**Source Code Verification**) und ausführbare Software (**Code Verification**).





(a) Einordnung des Sicherheitsmikrokosmos in die Entwicklung (nach [Pop05])



(b) Instanz des Musters Verifikationsschleife für Zugriffssicherheit

Abbildung 5.7: NFE-spezifische Anpassung des Musters Verifikationsschleife an Sicherheit

Auf den einzelnen Ebenen werden unterschiedliche Aspekte, die im Zusammenhang mit der Antwortzeit stehen, analysiert. So wird auf der PIM-Ebene überprüft, ob die auf einzelne Methoden dekomponierten lokalen Antwortzeiten in der Summe die globale Anforderung nicht überschreiten. Auf der PSM-Ebene wird anhand der plattformspezifischen Informationen eine Scheduling-Analyse durchgeführt. Auf der Code-Ebene werden durch eine WCET-Analyse die angenommenen Ausführungszeiten überprüft.

### Sicherheits-spezifische Bausteine am Beispiel von Zugriffssicherheit

Im Bereich von Sicherheitsanforderungen kann das NFE-Prozessmuster Verifikationsschleife beispielhaft auf den in [Pop05] definierten *Sicherheitsmikroprozess* abgebildet werden (Kapitel 4.1). Der Sicherheitsmikroprozess dient speziell der Überprüfung der vorab modellierten Anforderungen an die Zugriffssicherheit. Eingesetzt wird dieser in drei Phasen der Entwicklung (Abbildung 5.7(a)). Innerhalb der Geschäftsprozessmodellierung, der Anwendungsfallmodellierung und des Architektur Entwurfs (in [Pop05] als Analyse bezeichnet) wird der in Abbildung 5.7(b) dargestellte Ablauf zur Überprüfung der modellierten Maßnahmen gegen die spezifizierten Bedrohungen integriert. Dieser Ablauf ist eine Instanz des NFE-Prozessmusters Verifikationsschleife<sup>3</sup>.

Bei den Sicherheitsaspekten gibt es nur eine mögliche Instanz des Musters Verifikationsschleife, während es bei Realzeitaspekten (5.6) vier Instanzen gab. Popp fordert die Umsetzung der Überprüfung der Maßnahmen zur Sicherung der Zugriffssicherheit

<sup>3</sup>Da die Aktivitäten in [Pop05] auf deutsch definiert sind, wurden diese übernommen.

durch informale, semiformale oder formale Techniken ([Pop05], S.81). Deren konkrete Beschreibung ist offen gelassen. Eine weitere Aufsplittung in verschiedene Ausprägungen des Musters und damit verschiedene Techniken und deren Beschreibung der Integrationsmöglichkeiten in ein Vorgehensmodell ist durch den flexiblen Aufbau des PROKRIS-Ansatzes jedoch jederzeit möglich.

### 5.4.3 Ressourcenmodell

Ziel des PROKRIS-Frameworks ist es, die Prozessspezifikation sowohl als Dokumentation als auch als Steuerung für die notwendigen Arbeiten innerhalb einer Werkzeugumgebung zu nutzen. Ein Problem der Vorgehensmodellanpassung ist, dass zur Zeit der Modellierung des NFE-spezifischen Vorgehens die konkreten Ressourcen oftmals noch nicht bekannt sind. Die Methodik unterstützt daher ein wiederverwendbares *Ressourcenmodell*.

Ziel der Definition des eigenständigen Ressourcenmodells ist es, zur Modellierungszeit nur abstrakte Beschreibungen der Ressourcen innerhalb des Vorgehensmodells zu binden. Ein Beispiel dafür ist die Zuordnung, welche Rolle eine bestimmte Aufgabe übernehmen soll. Welche Person zur Zeit der Ausführung welcher Rolle zugeordnet ist, bleibt zu diesem Zeitpunkt offen. Es bedarf demnach der Unterscheidung von zwei Arten von Ressourcen, den *abstrakten Ressourcentypen* und den *konkreten, zur Laufzeit verfügbaren Ressourcen*, welche während der Ausführung verwendet werden können. Die abstrakten Ressourcen werden innerhalb des Verfeinerungsschrittes der Individualisierung auf konkrete Individuen bzw. konkrete Werkzeuge (z.B. allgemeines UML Werkzeug auf konkretes Werkzeug „TopCased“) abgebildet.

Abstrakte Ressourcen müssen zur Modellierungszeit bekannt sein. Dazu werden innerhalb des Prozesses Ressourcenschnittstellen definiert, an die zur Laufzeit des Prozesses konkrete Ressourcen gebunden werden können. Die konkreten Ressourcen, welche innerhalb einer Organisation zur Verfügung stehen, werden unabhängig in einem Ressourcenmodell definiert. Sie können so für verschiedene Vorgehensmodelle eingesetzt werden, sind also wiederverwendbar.

Die konkreten Ressourcen werden unterteilt in Personen, Programme und Daten. Modelliert werden dazu spezifische Daten, beispielsweise der Ausführungspfad und -parameter bei Programmen. Abbildung 5.8 zeigt ein Beispiel für eine konkrete Ressourcenspezifikation. Auf der linken Seite ist die Spezifikation im Ressourceneditor der PROKRIS-Bibliothek geöffnet, rechts sind Auszüge des zugehörigen XML-Codes dargestellt. Die Ressourcen werden in zwei Formen organisiert. Zum Einen ist im unteren Teil eine Liste der verfügbaren Ressourcen zu sehen, die nach Typen (**Programs**, **Data**, **Persons**) sortiert aufgeführt und durch zusätzliche Einträge genauer spezifiziert sind. Die Abbildung zeigt auf der rechten Seite beispielhaft den Eintrag für das Textverarbeitungswerkzeug *Notepad*. Die zweite Organisationsform ist die Gruppierung dieser Ressourcen innerhalb eines Ressourcenbaums (oberer Teil). Diese Gruppierung beschreibt den Verwendungszweck der Ressource. Beispielsweise werden in der Anforderungsanalyse (**Requirements**) *Use-Case*-Modelle mittels eines Editors (*Notepad*) bearbeitet. Die Zuordnung zu einer Gruppe erfolgt durch Referenzierung einer eindeutigen ID, welche innerhalb des Listenteils

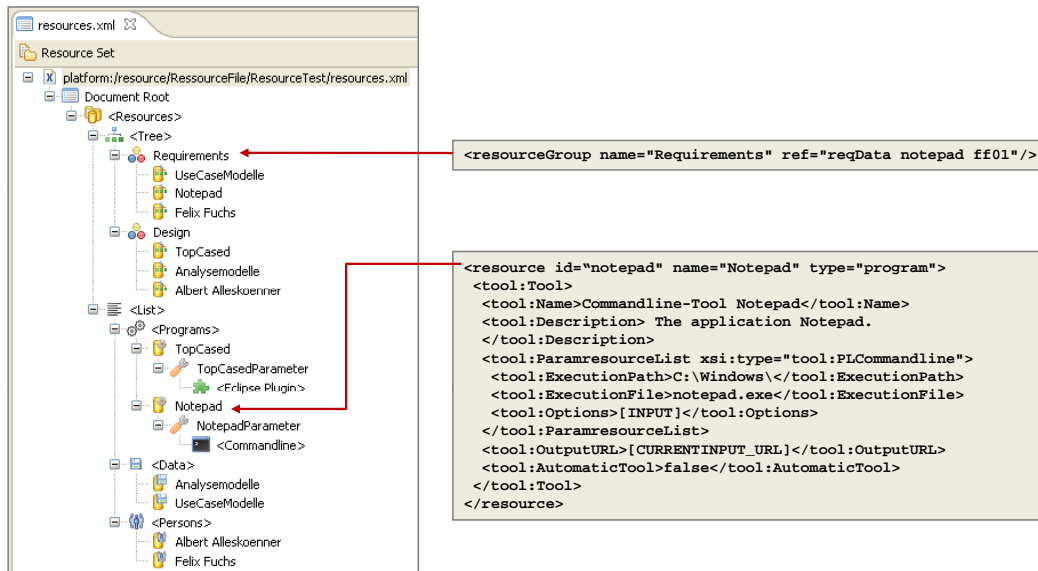


Abbildung 5.8: Beispiel für eine Ressourcenspezifikation

spezifiziert wird. Anhang C enthält ein ausführliches Beispiel einer Ressourcenspezifikation.

## 5.5 Kontexte und Facettenklassifikation

Die Spezifikation der NFE-Prozessmuster findet auf der Ebene der abstrakten NFE-Vorgehensmodelle statt. Die Bereitstellung der konkreten Verfahren erfolgt dagegen auf der Konkretisierungsebene der spezifischen NFE-Vorgehensmodelle durch NFE-spezifische Prozessbausteine. Wie das Beispiel in Abbildung 5.9 zeigt, erfolgt diese Instanziierung differenziert für verschiedene nicht-funktionale Anforderungen an das zu entwickelnde System. Es existiert also eine 1:n-Beziehung zwischen den Prozessmustern der abstrakten Ebene und den Prozessbausteinen der NFE-spezifischen Ebene (den Musterinstanzen). Innerhalb des nächsten Schrittes der Modellanpassung können die NFE-spezifischen Bausteine an die Organisationsstruktur einer Firma angepasst werden. Es ergibt sich dabei wiederum eine 1:n-Beziehung. Das bedeutet, dass innerhalb der Prozessbibliothek eine komplexe Struktur aus NFE-Prozessmustern bzw. -bausteinen entsteht. Diese wird als *NFE-Prozessbausteinlandschaft* bezeichnet.

Um die effiziente Arbeit mit der PROKRIS-Prozessbibliothek sicherzustellen, ist eine Strukturierung der Prozessbausteinlandschaft notwendig. Diese muss anhand unterschiedlicher Kriterien erfolgen, da in die Auswahl der Bausteine sowohl verschiedene Aspekte der NFE als auch der Entwicklungsphase und Granularität des Prozessbausteins eingehen. Die Strukturierung erfolgt daher durch die Einordnung der NFE-spezifischen Prozessbausteine in eine Facettenklassifikation ([Ran67], [Pri08]). Diese ermöglicht die Klassifikation von Sachverhalten nach mehreren kombinierbaren Gesichtspunkten, den

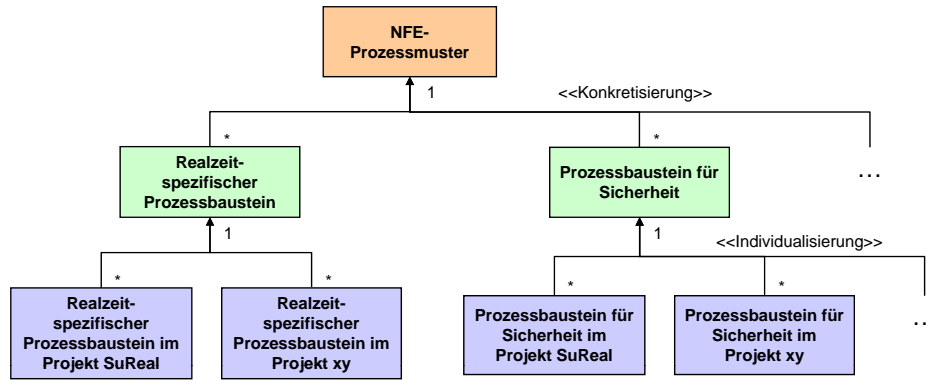


Abbildung 5.9: 1:n-Beziehungen der Bausteine zwischen den Modellanpassungsstufen

*Facetten.* Die Klassifizierung komplexer Sachverhalte unter Beachtung verschiedener Aspekte ist ein wesentlicher Vorteil gegenüber einer hierarchischen Klassifikation. Daher werden Facettenklassifikationen oft zur Strukturierung von Webseiten eingesetzt. Facettenklassifikationen bestehen aus mehreren Einfachklassen. Sie können außerdem durch Definition neuer Einfachklassen oder Facetten einfach erweitert werden.

Zur Strukturierung der PROKRIS-Bibliothek werden drei Facetten mit entsprechenden Unterklassen unterschieden. Ein Prozessbaustein kann dabei verschiedenen Ausprägungen der gleichen Facette zugeordnet werden, muss aber nicht zu jeder der drei Hauptfacetten eine Zuordnung besitzen. Die Zuordnung erfolgt grundsätzlich zu den Einfachklassen, welche den Blättern der Facetten entsprechen.

Im Folgenden werden die drei Facetten der PROKRIS-Klassifikation und deren Unterklassen eingeführt. Im Anschluss daran werden zwei konkrete Klassifikationen der PROKRIS-Bibliothek vorgestellt. Die Notation, der in den Beispielen verwendeten Beschreibungs- und Anfragesprache, beruht auf den Arbeiten von Priss ([Pri08], [Pri99]), in denen Beschreibungslogiken für Facettenklassifikationen verwendet werden.

### 5.5.1 Facetten der Klassifikation

Eine erste Facette wurde bereits in Kapitel 4.3 mit der Einteilung nach Ambler in verschiedene *Prozessmusterklassen* (*Process Pattern Type*) eingeführt. Für das Management der Prozessbausteine innerhalb der Prozessbibliothek ist außerdem eine detailliertere Klassifizierung notwendig, in die z.B. auch die Art der nicht-funktionalen Eigenschaften eingeht, für die ein Prozessbaustein eingesetzt werden kann. Die systematische Strukturierung nicht-funktionaler Eigenschaften bildet daher die Grundlage für eine weitere Dimension der Klassifizierung der Prozessbausteine innerhalb der Prozessbibliothek. Weiterhin ist eine Differenzierung der Phasen des Entwicklungsprozesses notwendig. Im Folgenden werden die beiden letzten Facetten der Klassifikation diskutiert.

### Facette der Entwicklungsphase

Nicht-funktionale Eigenschaften können in verschiedenen Detaillierungs- oder Abstraktionsgraden spezifiziert werden. Eine mögliche Unterscheidung ist folgende:

*Nutzerspezifische NFE*: Sowohl Käufer einer Software als auch Anwender stellen oft diffuse Anforderungen an die Qualität. Daher sind auf dieser Ebene oft metaphorisch Beschreibungen, wie „die Anwendung muss schnell sein“, zu finden.

*Anwendungsspezifische NFE*: sind Abbildungen der Nutzeranforderungen auf technische Parameter.

*Laufzeitspezifische NFE*: beschreiben benötigte Dienste der Plattform oder einer unterhalb der Software laufenden Middleware.

Diese Veränderung der Eigenschaften von hochgradig subjektiven Beschreibungen bis zu konkreten nicht-funktionalen Anforderungen verlangen innerhalb der einzelnen Entwicklungsphasen unterschiedliche Maßnahmen. Eine Möglichkeit der Einordnung der NFE-spezifischen Bausteine in die Facettenklassifikation ist die Ordnung nach den drei oben genannten Arten. Dabei wird vernachlässigt, dass die anwenderspezifischen NFE sowohl plattformunabhängige als auch plattformspezifische NFE umfassen. Da mit dem PROKRIS-Ansatz die Konzepte der MDA unterstützt werden sollen, werden diese nochmals unterteilt und folgende vier Unterklassen der Facette der *Entwicklungsphase* (*DevelopmentPhase*) eingeführt:

- *Requirements* — Behandlung nutzerspezifischer NFE
- *Architecture* — Behandlung plattformunabhängiger anwendungsspezifischer NFE
- *Design* — Behandlung plattform- und anwendungsspezifischer NFE
- *Implementation* — Behandlung laufzeitspezifischer NFE

### Facette der nicht-funktionalen Eigenschaften

Mit der Einordnung nicht-funktionaler Eigenschaften in eine Klassifikation beschäftigen sich bereits verschiedene Ansätze. In Kapitel 2.2 wurde eine Auswahl vorgestellt. Es existiert zwar kein allgemeingültiger Ansatz, der den Anspruch einer umfassenden Strukturierung aller nicht-funktionalen Eigenschaften erfüllen kann. Trotzdem bieten diese Ansätze einen guten Ausgangspunkt für die Struktur der PROKRIS-Prozessbibliothek.

Basierend auf den Ergebnissen der Analyse der vorgestellten Klassifikationen wird für die Facette *NFE* (*NFR*) eine Kombination aus den Ansätzen von Glinz und COMQUAD eingeführt. Die Facettenklassifikation von Glinz wird für die Oberklassen der Facette übernommen. Die Facette *NFR* (*NFE*) enthält damit folgende Klassen:

- *Kind* (Art) — mit dem vollständigen COMQUAD-Modell als Unterklassen
- *Representation* (Repräsentation) — mit den Unterklassen Operational (Operational), Quantitative (Quantitativ), Qualitative (Qualitativ) und Declarative (Deklarativ)

- *Satisfaction* (Erfüllung) — mit den Unterklassen *Hard* (Harte Anforderung) und *Soft* (Weiche Anforderung)
- *Role* (Rolle)<sup>4</sup> — mit den Unterklassen *Assumption* (Annahme) und *Fact* (Tatsache)

Die Klasse *Art* wird anhand der COMQUAD-Klassifikation weiter unterteilt. Die ausschließliche Unterscheidung aller nicht-funktionalen Eigenschaften in Laufzeit- und Entwicklungszeiteigenschaften gemäß Malan und Bredemeyer (Kapitel 1, [MB01]) wurde in dieser Klassifikation in der obersten Ebene integriert. Das PROKRIS-Framework schränkt sich auf Laufzeiteigenschaften ein. Bei Erweiterung der gesamten Methodik und des Frameworks auf Entwicklungszeiteigenschaften ist damit die problemlose Erweiterbarkeit der Klassifikation gewährleistet.

Die Facette *Rolle* unterscheidet im Gegensatz zu Glinz in Annahmen von NFE-Werten (*Assumption*) und bekannten, also gemessenen und bereits überprüften Werten (*Fact*).

### 5.5.2 Die PROKRIS-Klassifikation und zugeordnete Kontexte

Die einzelnen Facetten der Klassifikation werden verschiedenen *Kontexten* zugeordnet. Diese Kontexte repräsentieren Informationen, welche die Ausprägung des Vorgehensmodells beeinflussen<sup>5</sup>. Mittels der Kontextinformationen wird die Auswahl der Bausteine für eine bestimmte Abstraktionsstufe der Prozessspezifikation gesteuert. Das PROKRIS-Framework unterscheidet in *Anwendungs-*, *NFE-* und *Ausführungskontext* (*Application*, *NFR* und *Execution Context*). Außerdem dienen die Kontexte dazu, Abhängigkeiten der Prozessbausteine untereinander zu spezifizieren, um diese während der Modellanpassung auswerten zu können.

Die ersten beiden Kontexte sind auf Kriterien, welche aus den Anforderungen des Softwaresystems resultieren, ausgerichtet. Der Ausführungskontext beinhaltet dagegen Aspekte der Organisation. Um beide Arten nicht zu vermischen, werden in der PROKRIS-Methodik zwei Klassifikationen unterschieden. Beide Klassifikationen sind flexibel erweiterbar, so dass neue nicht-funktionale Eigenschaften oder andere Aspekte problemlos einbezogen werden können.

#### Facettenklassifikation von Anwendungs- und spezifischem NFE-Kontext

Diese Klassifikation besteht aus den drei diskutierten Facetten: *DevelopmentPhase*, *NFR* und *ProcessPatternType*. Letztere enthält die Unterklassen *Task*, *Activity* und *Phase* und beschreibt die Granularität der Prozessbausteine.

Abbildung 5.10 zeigt den Aufbau der gesamten Klassifikation. Zur Verdeutlichung der Nutzung der Klassifikation zeigt die Abbildung außerdem die Zuordnung des NFE-spezifischen Bausteins *Code Verification* (grün hinterlegt). Dieser wird folgendermaßen klassifiziert:

---

<sup>4</sup>Diese Rolle unterscheidet sich von den Rollen im Entwicklungsprozess. Die Bezeichnung wurde bewusst übernommen, um den Ansatz nach Glinz nicht zu verändern.

<sup>5</sup>Die Kontextinformationen repräsentieren demnach den Kontext eines Projekts und sind nicht zu verwechseln mit Kontexten in kontextabhängigen Anwendungen (context-aware computation).

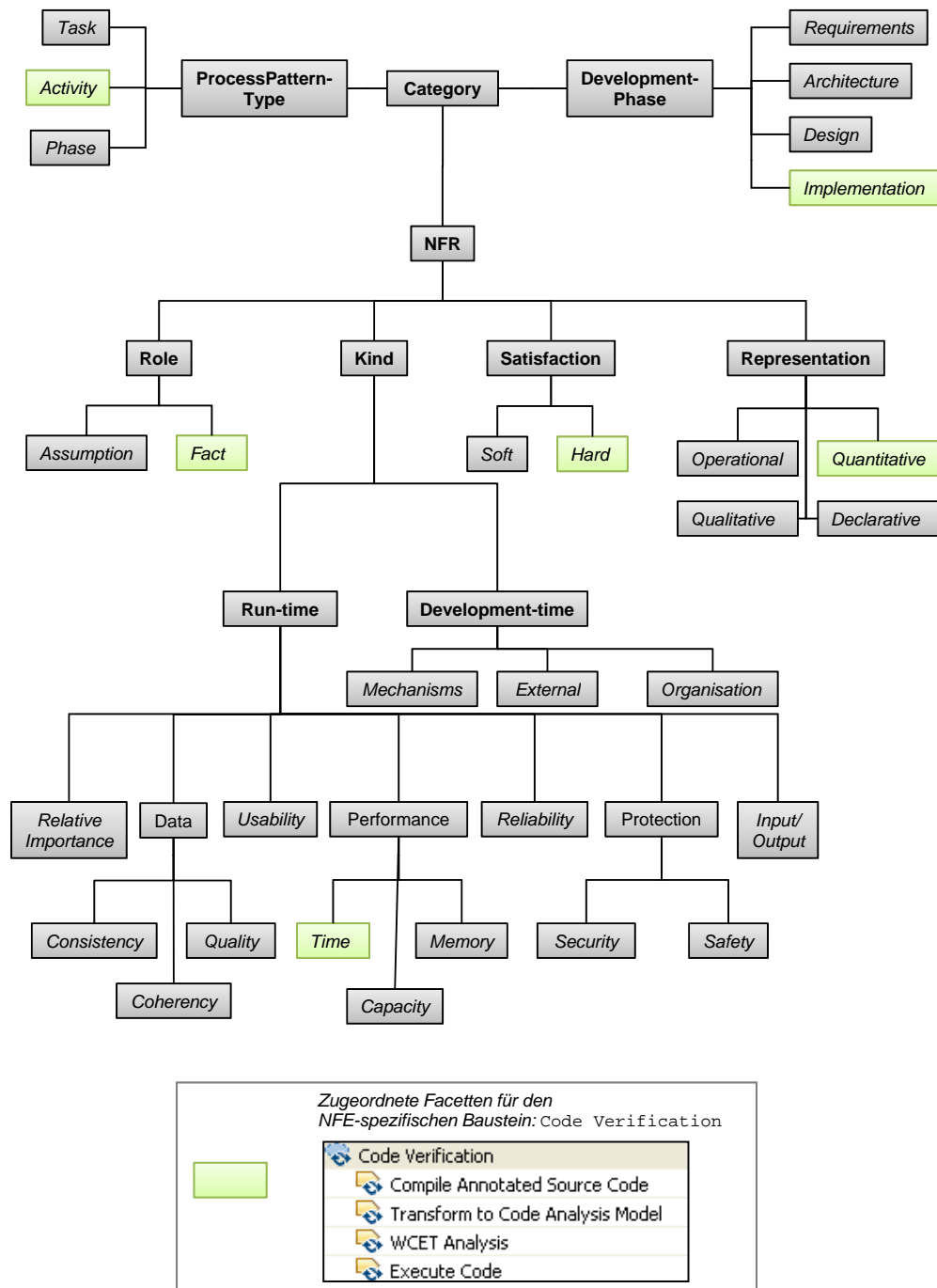


Abbildung 5.10: Struktur der PROKRIS-Facettenklassifikation

$$CodeVerification = Implementation \cap Activity \cap Role.Fact \cap Kind.Time \cap Satisfaction.Hard \cap Representation.Quantitative$$

### Facettenklassifikation des Ausführungskontexts

Der Ausführungskontext wird spezifisch für die jeweilige Organisation definiert. Er klassifiziert die konkreten Ressourcen nach bestimmten Merkmalen, wie Entwicklungsphase sowie Typ der Ressource. Abbildung 5.11 zeigt als Beispiel die Klassifikation der konkreten Ressourcen der Beispielspezifikation in Anhang C anhand des Ausführungskontexts. In dieser Spezifikation sind die Ressourcengruppen nach den Entwicklungsphasen klassifiziert. Das Modelchecker-Werkzeug „UPPAAL“ ist beispielsweise wie folgt eingeordnet:

$$UPPAAL = Development.PIM.verification \cap Program$$

## 5.6 Prozesskomposition

Der Begriff Prozesskomposition wird in dieser Arbeit historisch bedingt verwendet. Er wurde bereits in den ersten Veröffentlichungen zu Methoden- bzw. Prozessbibliotheken geprägt (z.B. in [HS03]). Der Begriff ist nicht zu verwechseln mit dem Kompositionsparadigma in der Softwareentwicklung ([BA01], [Szy02]). Dort wird meist von *Black-Box*-Komposition ausgegangen. Dabei werden die Komponenten ohne Veränderung ihrer inneren Struktur anhand einer Schnittstellenbeschreibung komponiert [Aßm03]. Der Begriff der Prozesskomposition beschreibt dagegen einen Filteralgorithmus, der es ermöglicht, geeignete Prozessbausteine zu finden und unnötige wegzuschneiden (Tailoring). Es existiert dabei keine abgeschlossene Prozesskomponente, die ihre Schnittstellen nach außen beschreibt. Prozesskomponenten sind im PROKRIS-Ansatz *White-Box*-Komponenten deren Inneres sichtbar und anpassbar ist. Deren Komposition erfolgt durch Kopieren und nachträgliches manuelles Editieren [Aßm03]<sup>6</sup>.

Die Prozesskomposition ist neben der Modellanpassung über die Verfeinerungsstufen eine der Hauptaktivitäten innerhalb des PROKRIS-Frameworks. Abbildung 5.12 zeigt eine Detailabbildung zu den Prozessmodellebenen. Wie zu sehen ist, bilden die Vorgehensbausteine den zentralen Kern, über die die Verfeinerung erfolgt. Die Erstellung der Vorgehensmodelle wird durch einen *Kompositionsfilter* unterstützt, welcher die Informationen der Kontexte, zu denen jedes Element der NFE-Bausteinlandschaft zugeordnet ist, zur Suche nach geeigneten NFE-Prozessbausteinen nutzt. Dabei werden für die unterschiedlichen Ebenen unterschiedliche Kontextarten ausgewertet. Über den Anwendungskontext werden die NFE-Prozessmuster gefiltert, über den NFE-Kontext die NFE-spezifischen Prozessbausteine und über den Ausführungskontext die verfügbaren konkreten Ressourcen.

Die Anpassung der Ressourcen erfolgt in zwei Stufen, der *statischen Bindung* zur Modellierungszeit und der *dynamischen Bindung* zur Laufzeit (Abbildung 5.12, unten).

<sup>6</sup>Eine Anwendung der *Black-Box*-Komposition ist ebenfalls denkbar, erfordert aber eine entsprechende Kompositionssprache für Vorgehens- bzw. Prozessmodellartefakte. Die Definition einer derartigen Sprache ist ein interessantes aber noch offenes Problem und sollte Gegenstand weiterer Forschung sein.



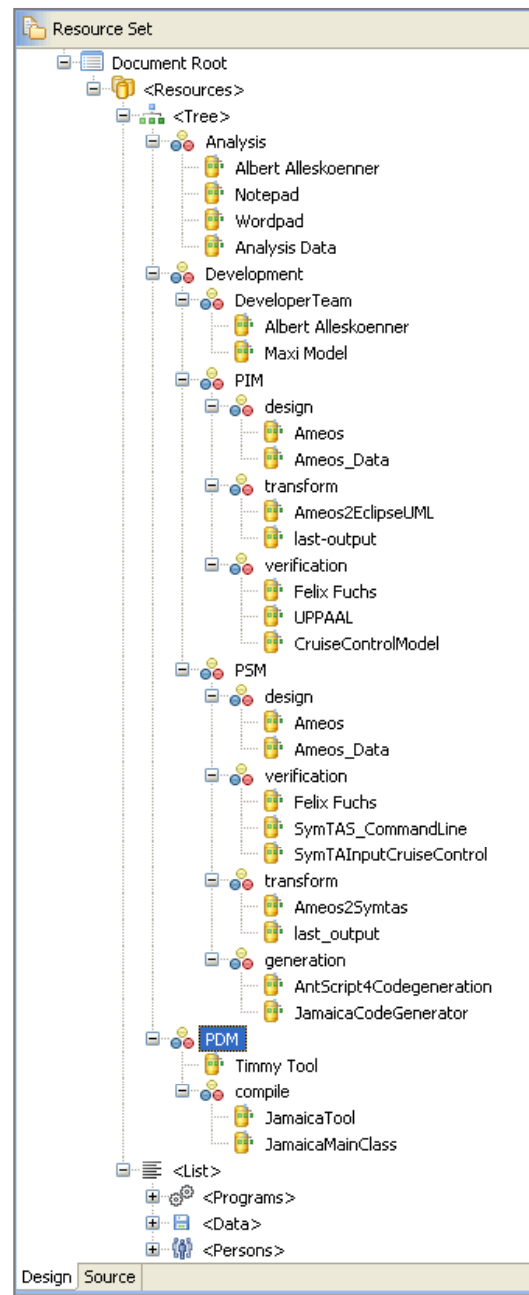


Abbildung 5.11: Klassifikation der konkreten Ressourcen des Beispiels in Anhang C anhand des Ausführungskontexts

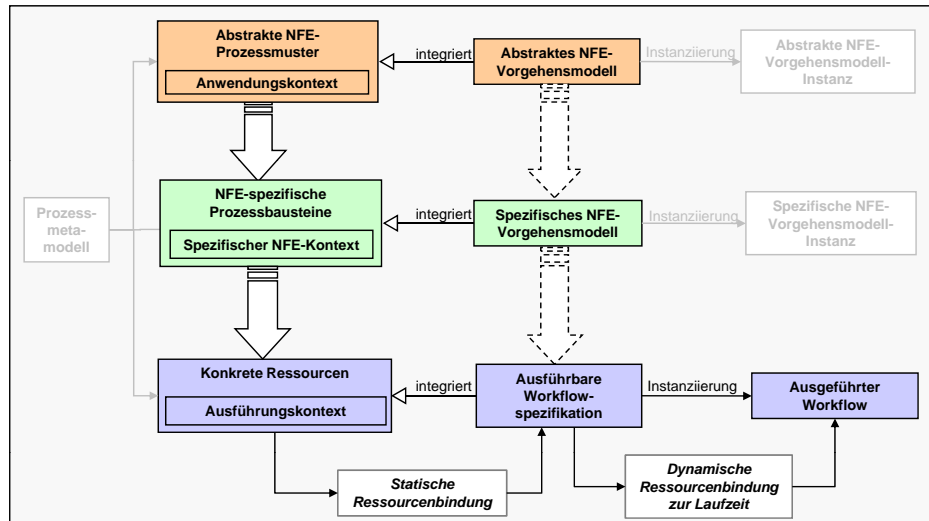


Abbildung 5.12: Ebenen der Komposition

Die Individualisierung übernimmt die statische Bindung durch Referenzen innerhalb der bereits beschriebenen Ressourcengruppen auf konkrete Einzelressourcen, welche zur Laufzeit dynamisch aufgelöst werden.

In der PROKRIS-Methodik können zwei Beziehungsarten zwischen Vorgehensmodell und wiederverwendbaren Prozesselementen unterschieden werden. Dies ist durch den Betrachtungspunkt auf die Prozessmodellierung bedingt. Auf den beiden oberen Ebenen arbeitet der Prozessingenieur mit grobgranularen Prozessbausteinen, welche Aktivitäten oder Phasen beschreiben. Man kann daher von einer „besteht aus“-Beziehung sprechen. Auf der letzten Stufe der PROKRIS-Methodik werden diese Komponenten durch den Projektmanager individualisiert, indem unter dem Einflussfaktor der Organisationsstruktur die Bausteine an die Ressourcen angepasst werden. Auf dieser Ausführungsebene wird feingranularer mit den einzelnen Aufgabendefinitionen gearbeitet, was durch den Einsatz von *White-Box*-Komponenten möglich ist. Auf dieser Ebene besteht damit eine „nutzt“-Beziehung.

## 5.7 Anwendung der PROKRIS-Methodik zur Komposition und Anpassung der erweiterten Vorgehensmodelle

Nachdem die einzelnen Konzepte erläutert wurden, soll nun deren Zusammenspiel zur Erstellung und Anpassung eines erweiterten Vorgehensmodells skizziert werden. Abbildung 5.13 gibt einen schematischen Überblick darüber. Nach der Erklärung der Zusammenhänge werden die einzelnen Anpassungsschritte anhand des Vorgehensmodells der SuReal-Fallstudie vertieft.

Abbildung 5.13 stellt sowohl die drei Stufen der NFE-Vorgehensmodelle als auch den Ablauf der kontextbasierten Anpassung dar. Das abstrakte NFE-Vorgehensmodell ent-

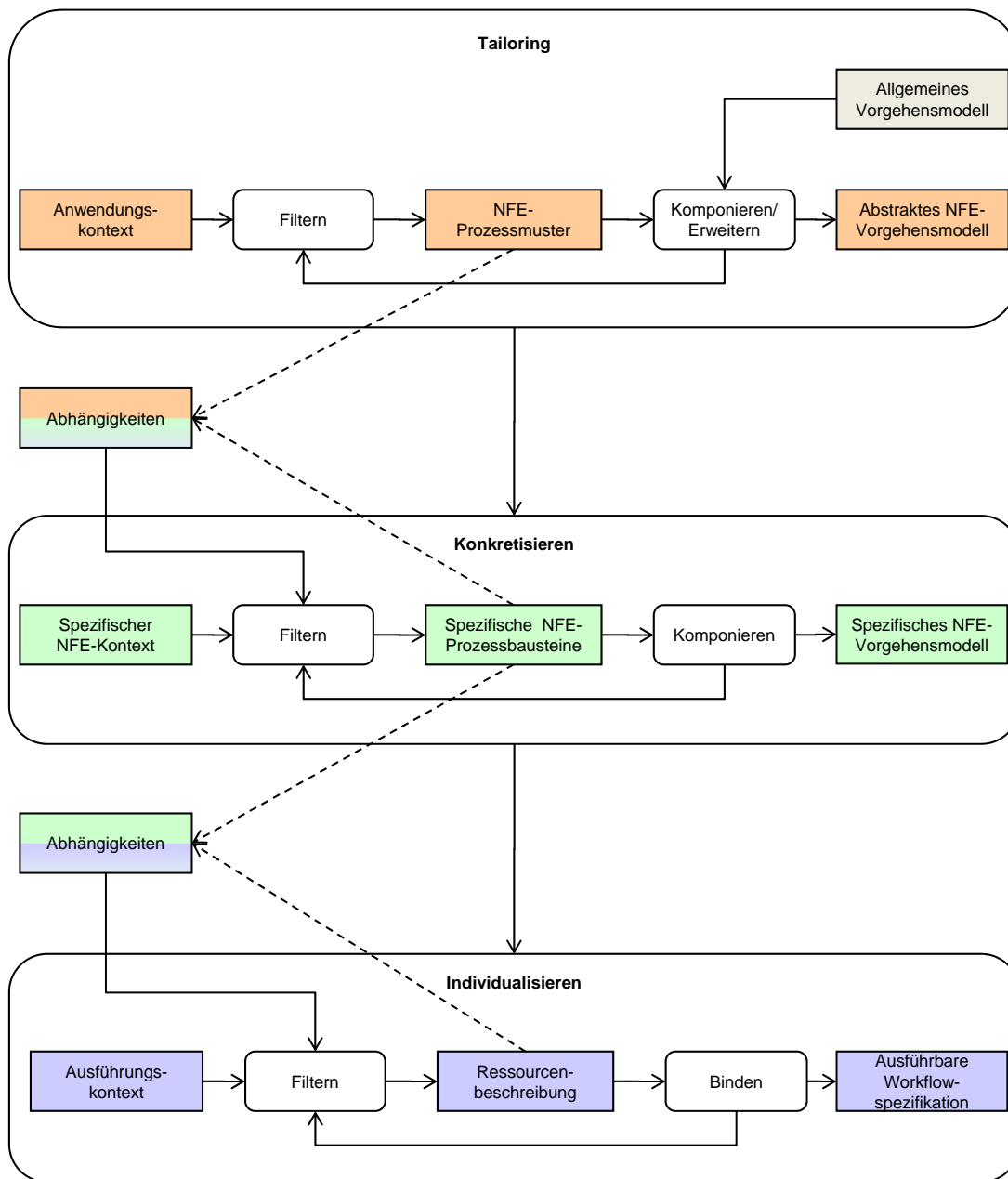


Abbildung 5.13: Schematische Darstellung der kontextabhängigen Anpassung

steht durch das Erweitern eines allgemeinen Vorgehensmodells. In diesem Schritt werden die notwendigen NFE-Prozessmuster ausgewählt und mit dem bestehenden Prozessmodell komponiert (Tailoring). Die Auswahl erfolgt über einen Filter, der als Kriterien allgemeine Anwendungskontexte (z.B. die Kritikalität des zu entwickelnden Systems) einsetzt. Das Erstellen des Modells erfolgt iterativ, d.h. es werden nicht alle Prozessmuster auf einmal integriert, sondern es ist ein schrittweises Erweitern des Modells möglich.

Im nächsten Schritt wird das abstrakte NFE-Vorgehensmodell durch Konkretisierung an die Umsetzung konkreter nicht-funktionaler Eigenschaften angepasst. Innerhalb dieses Schrittes werden iterativ die innerhalb der NFE-Prozessmuster definierten Methoden durch konkrete Techniken ersetzt. Die Techniken sind durch NFE-Prozessbausteine beschrieben. Anhand der NFE-spezifischen Kontexte, wie z.B. Art der NFE und Modellierungsebene, können durch den Prozessingenieur die entsprechenden Bausteine ausgewählt und so das Prozessmodell iterativ verfeinert werden. Außerdem werden spezielle Kontextinformationen ausgewertet, welche Abhängigkeiten zwischen verschiedenen Prozessbausteinen beschreiben. Dadurch entsteht das NFE-spezifische Vorgehensmodell, welches sowohl zur Dokumentation als auch zur Projektplanung eingesetzt werden kann.

Das NFE-spezifische Modell kann nun in einem weiteren Schritt an die organisationsspezifischen Ressourcen angepasst werden. Dazu existiert innerhalb des PROKRIS-Frameworks eine separate Ressourcenbeschreibung der verschiedenen Personen, Werkzeuge und Datenhaltungsmöglichkeiten. Anhand der organisatorischen Randbedingungen, die durch den Ausführungskontext abgebildet werden, können passende Einzelressourcen oder zusammengehörige Gruppen von Ressourcen ausgewählt und an die verschiedenen Aktivitäten gebunden werden. Durch diese Bindung entsteht eine individuell angepasste Prozessbeschreibung, die innerhalb der workflowunterstützten Werkzeugumgebung ausführbar ist.

### 5.7.1 Tailoring und Konkretisierung der statischen Modelle

Die beiden ersten Schritte der Vorgehensmodellanpassung behandeln statische Modelle. Nachdem kurz die dafür verwendeten Sprachkonzepte vorgestellt werden, wird auf die Anpassung eines Vorgehensmodells durch den Prozessingenieur am Beispiel der SuReal-Fallstudie eingegangen. Dazu werden die Werkzeuge des PROKRIS-Frameworks verwendet.

#### Sprachkonzepte

Die statischen Vorgehensmodelle werden unter Nutzung des UML-Profiles, basierend auf dem in dieser Arbeit erweiterten SPEM 2.0 Metamodell, modelliert. Den Kern eines Vorgehensmodells bildet dabei das Ablaufmodell (Work Break Down Structure). Abbildung 5.14 gibt einen Überblick über den Einsatz des Kompositionsfilters des PROKRIS-Frameworks. Im Dialogfenster unten rechts (5) ist ein Beispiel für ein Ablaufmodell zu sehen. Ein Ablaufmodell besteht im Wesentlichen aus Prozessmustern und Aktivitäten, welchen Rollen und Arbeitsprodukte zugeordnet sind. Ausführliche Beispiele für abstrakte und NFE-spezifische Modelle zeigt Anhang D.

## 5.7 Anwendung der PROKRIS-Methodik zur Komposition und Anpassung

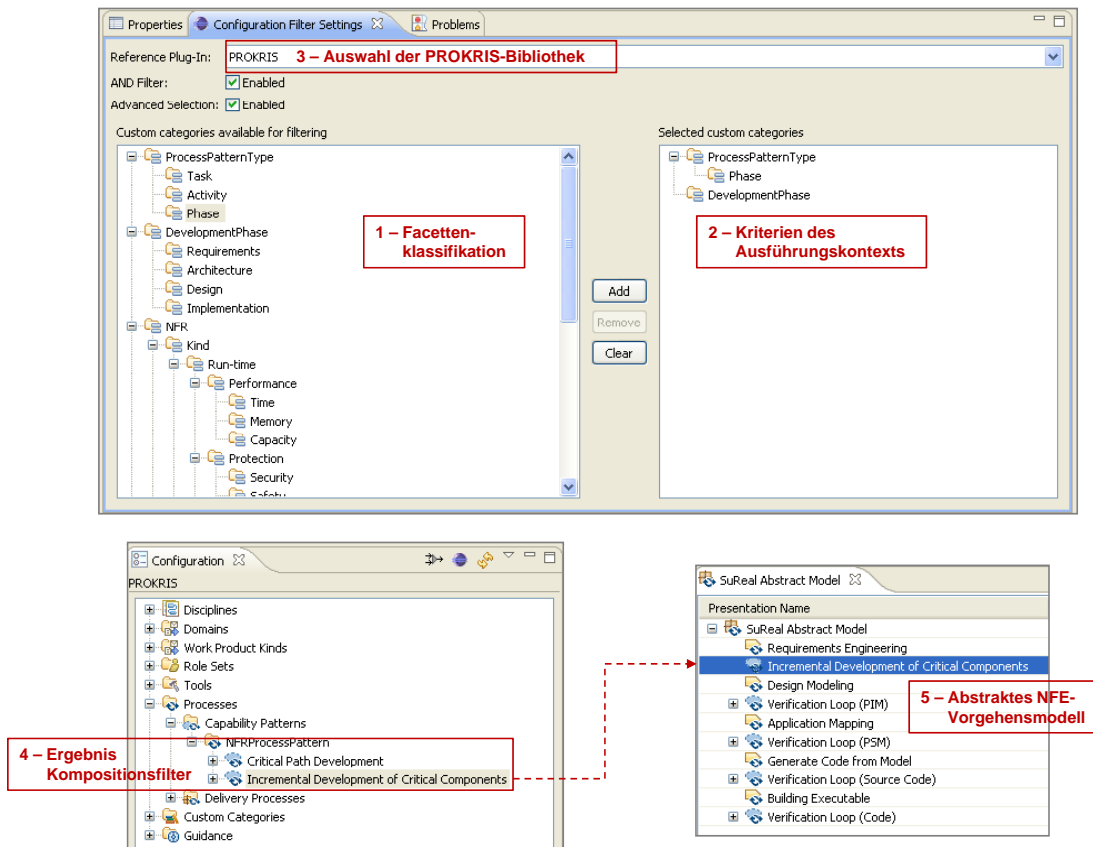


Abbildung 5.14: Beispiel für den Einsatz des PROKRIS-Kompositionsfilters zur Erstellung eines abstrakten NFE-Vorgehensmodells

### Tailoring der abstrakten NFE-Vorgehensmodelle

Der Schritt des Tailorings zur Erstellung des abstrakten NFE-Vorgehensmodells umfasst sowohl das Hinzufügen von NFE-Prozessmustern als auch das Anpassen bestehender Vorgehensmodelle.

Um in die Ablaufstruktur eines abstrakten NFE-Vorgehensmodells geeignete NFE-Prozessmuster einzufügen, wird der Kompositionsfilter genutzt. Während des Tailorings wird dieser so eingesetzt, dass er innerhalb eines eigenständigen Dialogs sowohl Filterkriterien in Form der Facetten als auch die Suchergebnisse anzeigt. Abbildung 5.14 zeigt die entsprechenden Ausschnitte aus dem Editor der PROKRIS-Prozessbibliothek. Das obere Fenster der Abbildung zeigt den Dialog zur Einstellung der Suchkriterien anhand des Anwendungskontexts. Dazu wird auf der linken Seite des Fensters die entsprechende Klassifikation angezeigt (1) und auf der rechten Seite die ausgewählten Kriterien (2).

Zum Erweitern des abstrakten NFE-Vorgehensmodells wird nicht die vollständige Klassifikation benötigt. Zum Anwendungskontext zählt die Facette **ProcessPatternType** (hier **Activity** ausgewählt) und die zusätzliche Unterscheidung von Mustern der Anwendungs- bzw. Domänenentwicklung. Durch Auswahl der Oberklasse **Development-**

**Phase** werden alle Muster der Anwendungsentwicklung berücksichtigt. Dass alle NFE-Prozessmuster zur Entwicklung laufzeitkritischer Eigenschaften einbezogen werden, ist durch den Einsatz der PROKRIS-Prozessbibliothek an sich gewährleistet. Dieser Punkt ist der dritte Aspekt des Anwendungskontexts und wird im Feld 3 ausgewählt.

Im linken unteren Fenster werden die Prozessmuster angezeigt, welche den festgelegten Kriterien des Anwendungskontexts entsprechen (4). Das Ergebnis:

$$X = (Critical\ Path\ Development, Incremental\ Development\ of\ Critical\ Components)$$

erfüllt folgende Anfrage:

$$X = NFR \cap DevelopmentPhase \cap ProcessPatternType.Phase$$

Die gefundenen Muster können nun in das abstrakte Vorgehensmodell integriert werden. Dabei ist zu beachten, dass es zwischen einzelnen NFE-Prozessmustern Beziehungen oder Abhängigkeiten geben kann. Diese werden in der jeweiligen Musterbeschreibung aufgeführt und diskutiert. Anhand dieser Beschreibung kann der Prozessingenieur entscheiden, welche weiteren NFE-Prozessmuster in das Vorgehensmodell zu integrieren sind. Eine automatische Auswertung dieser Beziehungen im Prototyp erfolgt derzeit nicht. Notwendig ist dazu eine Erweiterung der Klassifikation um die Relationen zwischen den NFE-Prozessmustern.

### Konkretisierung der spezifischen NFE-Vorgehensmodelle

Der Schritt der Konkretisierung erfolgt durch das Ersetzen der NFE-Prozessmuster durch spezifische Ausprägungen. Dabei kann der Prozessingenieur den Kompositionsfilter innerhalb des Ablaufmodells über einen Kontextmenüeintrag aufrufen. Die Auswahl der Kriterien erfolgt nun anhand des NFE-spezifischen Kontexts. Abbildung 5.15 zeigt dies im oberen Dialogfenster. Zum NFE-spezifischen Kontext gehören alle Facetten der vorgestellten Klassifikation. Außerdem muss der Filter die Instanzbeziehung zwischen den einzelnen NFE-Prozessmustern und deren NFE-spezifischen Ausprägungen auswerten. Das dargestellte Beispiel zeigt folgende Anfrage:

$$\begin{aligned} X = & (Refines \bullet Verification\ Loop) \\ & \cap NFR.Kind.Runtime.Performance.Time \cap NFR.Representation.Qualitative \\ & \cap NFR.Satisfaction.Hard \cap NFR.Role.Fact \\ & \cap DevelopmentPhase.Implementation \cap ProcessPatternType.Activity \end{aligned}$$

Die Ergebnisse werden dem Prozessingenieur entsprechend angeboten. Das Dialogfenster ähnelt dem Ergebnisfenster in Abbildung 5.14 und ist daher nicht dargestellt. Das Ergebnis für die Beispielanfrage ist:

## 5.7 Anwendung der PROKRIS-Methodik zur Komposition und Anpassung

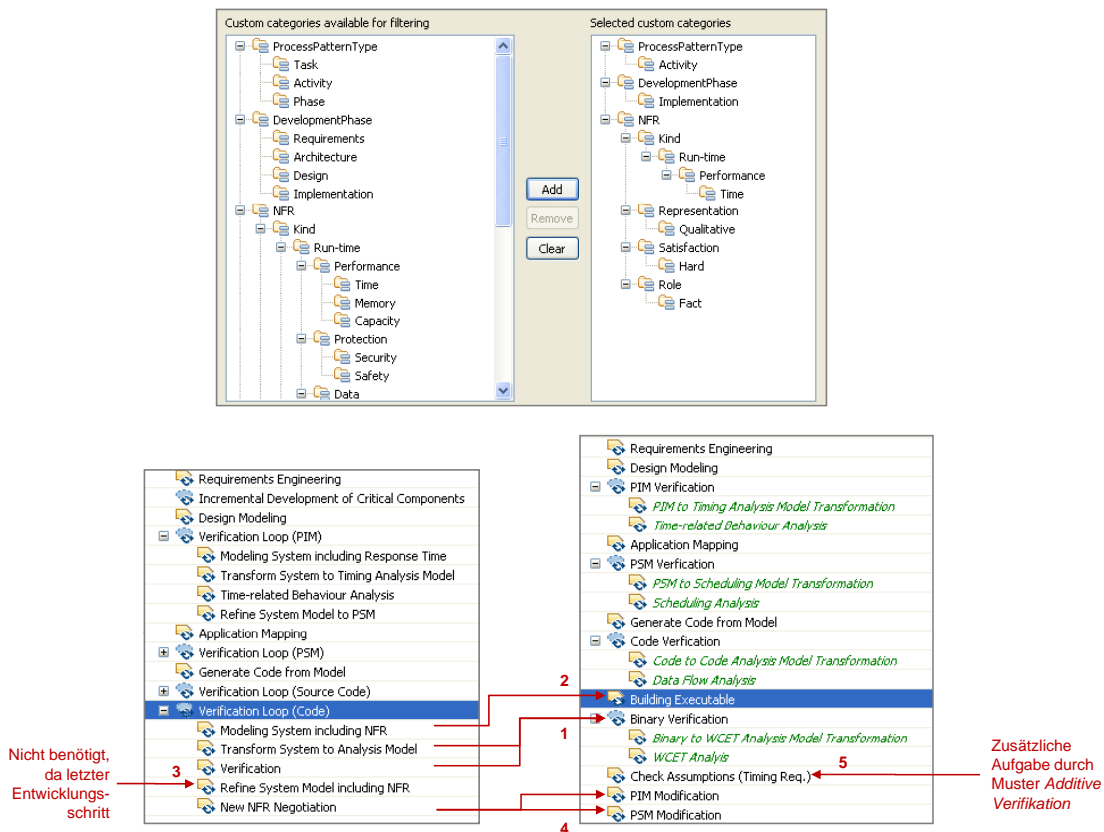


Abbildung 5.15: Beispiel für den Einsatz des Kompositionsfilters zur Konkretisierung des spezifischen NFE-Vorgehensmodells

Building Executable	14	11		Role Descriptor
Programmer			Primary Performer	Role Descriptor
Annotated RT-Java Code			Mandatory Input	Artifact Descriptor
Class Files			Mandatory Input	Artifact Descriptor
Class Files			Output	Artifact Descriptor
Executable Binary			Output	Artifact Descriptor
Binary Verification	15	14	extends 'binary_verification, surreal'	Capability Pattern
Binary to WCET Analysis Model Transformation	16			Task Descriptor
Verification Expert (Binary)			Primary Performer	Role Descriptor
Executable Binary			Mandatory Input	Artifact Descriptor
WCET Analysis Model			Output	Artifact Descriptor

Abbildung 5.16: Ausschnitt aus der zusammenfassenden Ansicht des realzeitspezifischen Vorgehensmodells

$$X = ((BinaryVerification \in NFESpecificBuildingBlock) , \\ (BuildingExecutable \in Activity))$$

Neben einer Einzelaktivität wird ein `NFESpecificBuildingBlock` gefunden. Dieser beinhaltet zwei Aktivitäten, welche die Hauptaktivitäten des zu konkretisierenden Prozessmusters an die Behandlung von Realzeitanforderungen auf Code-Ebene anpassen.

Abbildung 5.15 zeigt im unteren Bereich links einen Auszug aus dem abstrakten NFE-Vorgehensmodell des SuReal-Beispiels und rechts das realzeitspezifische NFE-Vorgehensmodell nach der Konkretisierung des Prozessmusters **Verification Loop** innerhalb der Bereitstellung des ausführbaren Codes. Dargestellt ist der iterative Ablauf des Konkretisierungsschritts. Die beschriebene Anfrage ergibt die Ergebnisse für die Iterationschritte 1 und 2 in Abbildung 5.15. Da es sich bei den NFE-Prozessmustern um White-Box-Komponenten handelt, ist es dem Prozessingenieur möglich, die Ersetzung des Musters durch die Ergebnisse der Anfrage nicht nur zu überprüfen, sondern zu beeinflussen. In diesem Fall wird die Aktivität **Refine System Model including NFR** nicht benötigt, da es sich um den letzten Entwicklungsschritt handelt und daher die NFE nicht weiter verfeinert werden (Iterationsschritt 3). Ein weiterer Iterationsschritt, den der Prozessingenieur durchführen muss, ist die Zuordnung der entsprechenden Aktivitäten im realzeitspezifischen NFE-Vorgehensmodell zur Aktivität **New NFR Negotiation** im abstrakten NFE-Vorgehensmodell (Iterationsschritt 4). Die Aktivität **Check Assumptions** (Iterationsschritt 5) ist nicht aus dem NFE-Prozessmuster *Verifikationsschleife* (**Verification Loop**) hervorgegangen, sondern wird durch das NFE-Prozessmuster *Additive Verifikation* hinzugefügt. Dieser Schritt verdeutlicht eine weitere Herausforderung für den Prozessingenieur während der Konkretisierung. Änderungen auf der Ebene der spezifischen NFE-Vorgehensmodelle können Änderungen auf der abstrakten Ebene bewirken. Daher ist es notwendig, die entstehenden NFE-Vorgehensmodelle auf ihre Konsistenz zu überprüfen. Abbildung 5.16 zeigt einen Ausschnitt aus der Ansicht des vollständigen realzeitspezifischen NFE-Vorgehensmodells. Zu überprüfen sind sowohl die Rollenzuordnungen, aber auch der Datenfluss der Arbeitsprodukte. Hier dargestellt ist das Arbeitsprodukt **Executable Binary**.



### 5.7.2 Individualisierung der ausführbaren Modelle

Die Individualisierung der ausführbaren Prozessmodelle ist der dritte Schritt in der Anpassung der erweiterten Vorgehensmodelle. Diese liegen nach erfolgter Übersetzung in der Ausführungssprache aPDL vor, deren notwendige Konzepte kurz erläutert werden. Weiterhin wird die statische Ressourcenbindung diskutiert. Die dynamische Bindung erfolgt durch die Ausführungsumgebung und wird dort behandelt (Kapitel 5.8).

#### Sprachkonzepte

aPDL erweitert die Sprache jPDL, deren Grundlagen in Kapitel 2.8 eingeführt wurden, um Ressourcenschnittstellen. Listing 5.1 zeigt als Beispiel die Definition einer Aufgabe (<task-node>), wie sie nach der Übersetzung aus dem NFE-spezifischen Model vorliegt. Die Ressourcenschnittstellen (<resourceInterface>, Zeile 5) beschreiben die Ankerpunkte, an die konkrete Ressourcen über die statische Ressourcenbindung gebunden werden. Sie sind als <action> (siehe Kapitel 2.8) innerhalb des Ereignisses <task-start> modelliert. So wird gewährleistet, dass während der Laufzeit beim Betreten der Aktivität die dynamische Ressourcenbindung erfolgt, deren Logik in der Klasse ResourceAssignmentHandler implementiert ist (Zeilen 3, 4).

Listing 5.1: Beschreibung einer Aufgabe mit abstrakten Ressourcen in aPDL

```

1 <task-node name="Model PIM with Topcased">
  <task name="Model PIM (Topcased)" description="Use UML diagrams like Class, State
    and Sequence Diagrams to describe the design of the system.
    ##surreal\deliveryprocesses\design_modeling_91B68291.html">
3   <event type="task-start">
4     <action class="org.surreal.projekt.eclipse.spms.handler.
      ResourceAssignmentHandler" name="Resource binding">
5       <resourceInterface>
6         <role abstract="Developer" />
7         <workproduct abstract="PIM Model" />
8         <tool abstract="UML Tool" />
9       </resourceInterface>
10    </action>
11  </event>
12 </task>
13 <transition to="Convert to UPPAAL and verify"></transition>
</task-node>

```

#### Statische Ressourcenbindung

Innerhalb der NFE-spezifischen Vorgehensmodelle sind abstrakte Ressourcen in Form von Rollen (Role), Arbeitsproduktbeschreibungen (WorkProduct) und Werkzeugbeschreibungen (Tool) gebunden. Diese werden als abstrakte Ressourceneinträge übersetzt (Listing 5.1, Zeile 6-8) und während der Individualisierung auf vorhandene konkrete Ressourcen der Organisation abgebildet. Die Individualisierung übernimmt damit die statische Bindung durch Referenzen innerhalb der bereits beschriebenen Ressourcengruppen auf konkrete Einzelressourcen, welche zur Laufzeit dynamisch aufgelöst werden.

Listing 5.2 zeigt ein Beispiel für die Bindung von Ressourcengruppen. Innerhalb der Ressourcenschnittstelle sind nun Gruppen konkreter Ressourcen gebunden.

Listing 5.2: Ressourcenschnittstelle mit Gruppen konkreter Ressourcen

```

...
2  <resourceInterface>
    <role group="Development.DeveloperTeam" />
4  <workproduct group="Development.PIM.Design" />
    <tool group="Development.PIM.Design" />
6  </resourceInterface>
..

```

Als Kriterium für die Auswahl der konkreten Ressourcen dient der Ausführungskontext. Dieser ist organisationsspezifisch. Für das Beispiel sind, unter Nutzung der vorgestellten Beispielklassifikation des Ausführungskontexts, folgende Anfragen an die PROKRIS-Prozessbibliothek notwendig:

$$\begin{aligned}
 \text{role group} &= \text{Development} \cap (\text{Resource.Role} \circ \text{Developer}) \\
 \text{workproduct group} &= \text{Development} \cap (\text{Resource.Workproduct} \circ \text{PIM}) \\
 \text{tool group} &= \text{Development} \cap (\text{Resource.Tool} \circ \text{UMLTool})
 \end{aligned}$$

Durch die Individualisierung werden die Aufgaben auf verschiedene Personengruppen verteilt. Außerdem wird sowohl die konkrete Werkzeugkette als auch der Datenfluss der Prozessprodukte modelliert. Betrachtet wird in dieser Arbeit eine rein syntaktische Komposition. Das bedeutet, die Auswahl erfolgt unter der Maßgabe einer eindeutigen Bezeichnung der konkreten Ressourcen und deren sinnvollen Zuordnung zu den abstrakten Ressourcen.

Während der Individualisierung entstehen Werkzeugketten, innerhalb denen automatische und manuelle Werkzeuge unterschieden werden. Automatische Werkzeuge benötigen keine Nutzerinteraktionen und können daher während der Laufzeit automatisiert ablaufen. Die Anpassung der Nutzungsdaten zwischen verschiedenen Datenformaten der Werkzeuge erfolgt über Adapter, welche als eigenständiges Werkzeug die Datentransformationen übernehmen. Da die Datenmodellierung nicht Kernthema dieser Arbeit ist, wird hier nicht weiter darauf eingegangen. Prinzipiell gibt es bei der datenbasierten Werkzeugintegration interessante Ansätze, deren Einbindung in den PROKRIS-Ansatz weiter verfolgt werden sollte. Eine kurze Diskussion wird dazu in Kapitel 11 innerhalb der Betrachtung weiterführender Forschungsaufgaben geführt.

## 5.8 Ausführungsunterstützung innerhalb eines Projekts

Ein weiteres Ziel der Methodik ist die Unterstützung der Ausführung der Prozesse. Die zugrundeliegende Idee ist die Nutzung der Prozessspezifikationen während der laufenden Entwicklung sowohl als *Dokumentation* als auch als *Steuerung* für die notwendigen Arbeiten *innerhalb einer prozessbasierten Werkzeugumgebung*. Abbildung 5.17 zeigt im Detail die Instanzebene (M0) des Frameworks. Zu sehen ist die Unterstützung des Exports in ein webbasiertes Nutzerhandbuch (Dokumentation) sowie der Export in eine ausführbare Spezifikation.

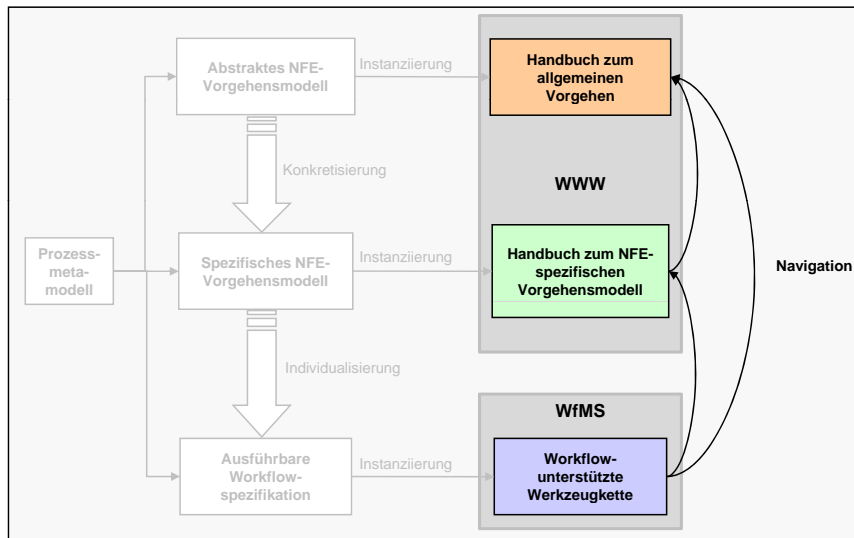


Abbildung 5.17: Ausführungsunterstützung des PROKRIS-Frameworks

Die prozessbasierte Werkzeugumgebung baut auf einer Workflowmaschine auf. Dem Nutzer werden über eine entsprechende Nutzerschnittstelle die prozessrelevanten Informationen angezeigt. Über diese Schnittstelle kann er Aufgaben mit den zugehörigen Werkzeugen und Daten bearbeiten und sich über das integrierte Handbuch Hilfe holen.

### Arbeitssteuerung durch prozessbasierte Werkzeugumgebung

Die ausführbare Workflowspezifikation kann durch eine erweiterte *Workflowmaschine* (*PROKRIS-Ausführungsumgebung*) interpretiert und ausgeführt werden. Durch Auswertung der Spezifikation, insbesondere des Ressourcenmodells, ist es möglich eine individuell angepasste integrierte Werkzeugumgebung bereitzustellen, die durch einen Prozess unterlegt ist.

Der Entwickler sieht seine Aufgaben und kann aus der Umgebung heraus die entsprechenden Werkzeuge starten. Eine Unterstützung des kollaborativen Arbeitens in verteilten Entwicklerteams ist ebenso denkbar wie die bessere Verfolgbarkeit von Anforderungen durch die Überwachung globaler nicht-funktionaler Anforderungen bei der parallelen Entwicklung von zusammenspielenden Komponenten. Die Ausführungsumgebung kann dabei gezielt an die Anforderungen des Projekts und seines Teams angepasst werden. So ist es durchaus möglich, den gesamten Projektablauf zu unterstützen. Im Hinblick auf die bereits geführte Diskussion zur Agilität versus Formalität von Prozessmodellen ist es ebenso möglich, den Einsatz der prozessbasierten Entwicklungsumgebung auf die laufzeitkritischen Teile der zu entwickelnden Software zu beschränken. Dabei wird die Entwicklung der kritischen NFE-lastigen Komponenten durch einen formalen Prozess geführt. In unkritischen Bereichen stehen dem Entwickler weiterhin Best Practice Methoden zur freien Wahl zur Verfügung.

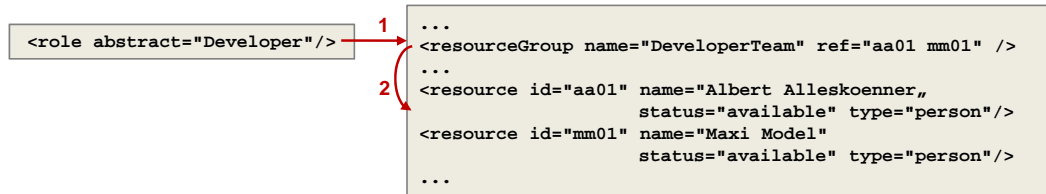


Abbildung 5.18: Zweistufige Ressourcenbindung an einem Beispiel

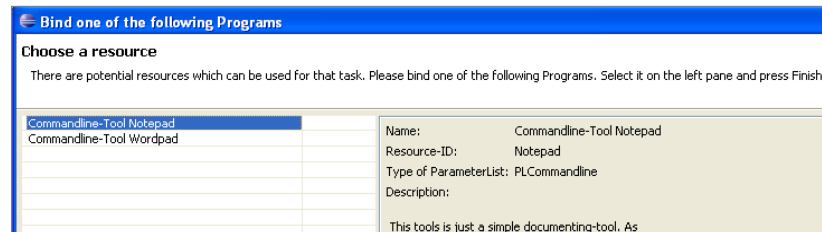


Abbildung 5.19: Beispiel für einen Dialog zur Auswahl eines Werkzeugs aus einer Ressourcengruppe

### Dynamische Ressourcenbindung zur Laufzeit

Durch die gewählte Form des extern vom Prozessmodell spezifizierten Ressourcenmodells wird es möglich, die konkreten Ressourcen erst zur Laufzeit des Prozesses zu binden, indem aus der jeweils gebunden Ressourcengruppe zur Abarbeitungszeit eine konkrete Ressource ausgewählt wird. So erreicht man eine hohe Flexibilität in Bezug auf die Verwendung der Ressourcen und kann zum letztmöglichen Zeitpunkt entscheiden, welche reale Ressource genutzt wird.

Zur Ausführungszeit der Prozessspezifikation werden die Referenzen innerhalb der Ressourcengruppen aufgelöst. Abbildung 5.18 zeigt ein Beispiel. Schritt 1 stellt die statische Ressourcenbindung dar. Speziell wird hier eine Gruppe von zwei Entwicklern gebunden. Zur Laufzeit kann einer der beiden Entwickler die Aufgabe übernehmen. Erst dann wird dieser Entwickler bzw. seine Ressourcendaten fest an die Aktivität gebunden (Schritt 2). Dieser Schritt wird als dynamische Ressourcenbindung bezeichnet. In der Nutzeroberfläche ist die Auswahl durch eindeutige Zuordnung zur Aufgabenliste des Entwicklers zu visualisieren.

In der selben Weise werden Werkzeuge und Daten gebunden. Dem Nutzer wird ein entsprechendes Auswahlfenster angeboten und nach erfolgter Zuordnung das ausgewählte Werkzeug mit den Daten gestartet. Abbildung 5.19 zeigt beispielhaft ein entsprechendes Dialogfenster.

Die dynamische Bindung geschieht zum Zeitpunkt der Ausführung einer Aktivität beim Aufruf von `<action ... name=Resource binding"/>` (Listing 5.1, Zeile 4). Dadurch ist die Spezifikation der Einzelressourcen während der Projektlaufzeit unabhängig von der Prozessspezifikation flexibel veränderbar.

### Dokumentation durch webbasiertes Nutzerhandbuch

Durch die Anbindung der Ausführungsumgebung an die PROKRIS-Prozessbibliothek stehen weiterführende Informationen zu Methoden, Techniken, Rollen, Produkten und Werkzeugen zur Verfügung. Diese können zusammen mit der Ablaufstruktur in ein webbasiertes Nutzerhandbuch exportiert werden. Der Vorteil gegenüber einem Papierhandbuch liegt auf der Hand. Änderungen können leichter eingepflegt werden. Außerdem kann die Dokumentation mit der PROKRIS-Ausführungsumgebung verbunden werden. So sieht der Entwickler nicht nur seine Aufgaben, er erhält außerdem aufgabenspezifische Hilfen durch die Einbindung des Nutzerhandbuchs in die Umgebung.

Die Voraussetzung für diese Verbindung wird durch die Modellierung der Verweise zu den entsprechenden Einträgen des Handbuchs innerhalb der ausführbaren Modelle geschaffen. Listing 5.1 zeigt ein Beispiel für die Spezifikation der entsprechenden Informationen. Zu sehen ist eine Aufgabe `<task>` (Zeile 2) deren Attribut `description` im ersten Teil eine kurze Beschreibung der Aufgabe enthält. Im zweiten Teil ist die relative Angabe eines Verweises auf die entsprechende Seite des webbasierten Handbuchs zu sehen. In der Oberfläche der Ausführungsumgebung kann so zu jeder Aufgabe sowohl eine kurze Beschreibung als auch der Verweis zum Handbucheintrag der entsprechenden Aufgabe zur Verfügung gestellt werden. Durch die webbasierte Bereitstellung des Handbuchs kann der Nutzer von diesem Eintrag aus zu relevanten Erläuterungen und Hilfen navigieren.

Die Struktur wird durch den gleichzeitigen Export des aPDL-Codes und des Handbuchs aus dem in SPEM modellierten Vorgehensmodell erzeugt.

## 5.9 Subprozesse - Parallelität in der Entwicklung

### Subprozesse durch Architekturentscheidungen

Bisher wurde ein Vorgehensmodell als ein einzelner Pfad der Abarbeitung betrachtet, welcher durch prozessgetriebene Entscheidungen verfeinert und innerhalb eines konkreten Projekts ausführbar ist. Abbildung 5.20 stellt dies auf der rechten Seite unter Nutzung der eingeführten Termini dar.

Die Komposition eines geeigneten Entwicklungsprozesses ist nicht nur abhängig von den geforderten nicht-funktionalen Eigenschaften. Sie wird auch unmittelbar von Architekturentscheidungen beeinflusst. Diese bedingen eine Unterteilung des Systems in Subsysteme bzw. Komponenten. Damit verbunden ist eine Aufsplittung des Prozessverlaufes in einzelne Subprozesse für jedes der Teilsysteme. Aus Effizienz- und Kostengründen werden diese im Laufe eines Projekts von unterschiedlichen Teams bearbeitet. Der Ablauf der Subprozesse kann parallel ablaufen. Zur Unterstützung der innerhalb von PROKRIS betrachteten Systeme ist eine kontinuierliche Überprüfung des Gesamtsystems auf die Einhaltung der globalen nicht-funktionalen Anforderungen notwendig. Daher sind in der Prozessmodellierung *Synchronisationspunkte*, an denen die Komponenten zur Überprüfung des Gesamtsystems wieder zusammengeführt werden, vorzusehen. Abbildung 5.21 zeigt ein Beispiel für einen Subprozess innerhalb des SuReal-Vorgehensmodells. Die übliche Struktur des Prozesses ist leicht erkennbar, jedoch wird nach der Akti-

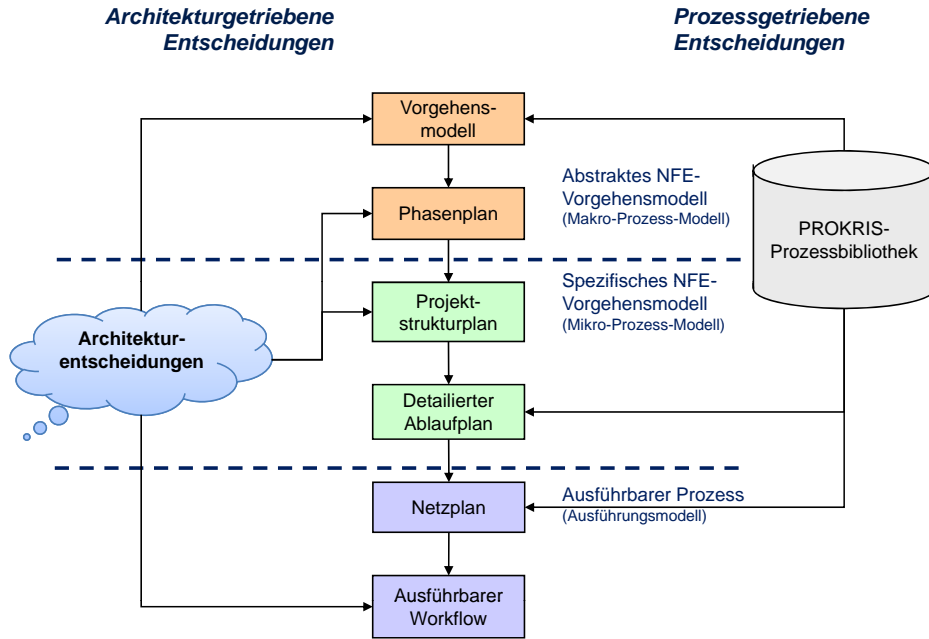


Abbildung 5.20: Prozess- versus architekturbasierte Entscheidungen

vität PIM Modellierung eine weitere Aufgabe **Aufteilung in Subsysteme (Component Splitting)** eingefügt. Diese realisiert das Aufsplitten des Systems in Komponenten und stößt für jede einen parallelen Prozessabschnitt an. Für jede der einzelnen Komponenten wird eine Verifikation der Echtzeitanforderungen durchgeführt und anschließend der Prozessablauf wieder zusammengeführt, um die Zeitprüfung für das gesamte System zu ermöglichen. Dies wird durch die Aktivität **Zusammenführung der Subsysteme (Component Synchronisation)** realisiert.

Der Einfluss der Architekturentscheidungen auf die Prozessunterstützung wirft neue Anforderungen auf, welche in der vorliegenden Arbeit erkannt wurden. Eine Lösung innerhalb des PROKRIS-Ansatzes erfordert umfangreiche weiterführende Arbeiten. Dazu zählen:

- Die Definition der Aufgaben von **Component Splitting** und **Component Synchronisation** im Kontext der nicht-funktionalen Eigenschaften, sowie deren Rahmenbedingungen.
- Die Definition möglicher paralleler Teilabschnitte innerhalb des erweiterten Vorgehensmodells sowie der notwendigen Synchronisationspunkte.
- Unterstützung der Verfolgbarkeit der nicht-funktionalen Eigenschaften transparent über die Subprozesse als Kommunikationsmöglichkeit zwischen den einzelnen Bearbeitern.
- Unterstützung einer verteilten Entwicklung der Subprozesse.

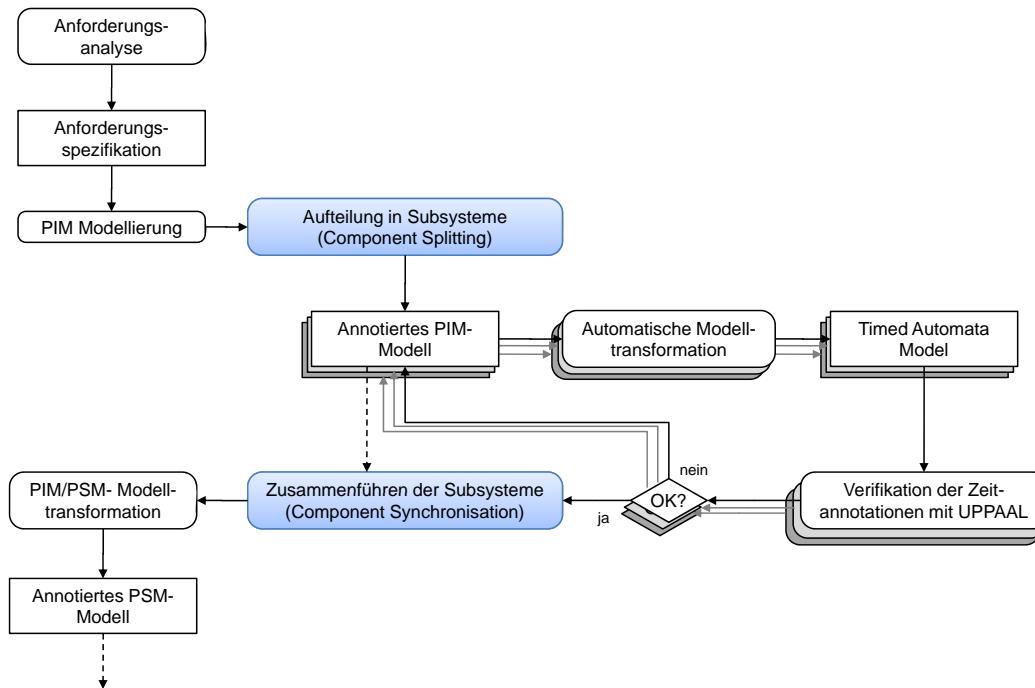


Abbildung 5.21: Beispiel für Subprozesse

Der letzte Punkt wird im PROKRIS-Ansatz bereits berücksichtigt, wurde aber auf den Einsatz für Subprozesse und deren Synchronisation noch nicht evaluiert.

### Behandlung auf den verschiedenen Abstraktionsebenen

Abbildung 5.20 ist zu entnehmen, dass die Architekturentscheidungen an verschiedenen Stellen im Zyklus der Prozessverfeinerung einen Einfluss haben. Während die Aufteilung des Systems auf der Makro- und Mikroprozessebene noch statisch ist, geschieht die Aufteilung während der laufenden Entwicklung dynamisch.

In der Makroebene (abstrakte NFE-Vorgehensmodelle) erfolgt eine grobe Aufteilung des Systems in seine verschiedenen Bestandteile. Diese können sowohl Hardware als auch Software sein. Die Entwicklung dieser Teile unterliegt meist verschiedenen Vorgehensmodellen. Es existieren daher auf dieser Ebene separate Prozessdefinitionen.

Auf der Mikroebene (spezifische NFE-Vorgehensmodelle) der Prozesse erfolgt die Betrachtung der Architektur und damit die Aufteilung in Subsysteme. Diese Aufteilung der Komponenten bleibt über den Entwicklungszeitraum konstant und kann durch die Modellierungskonzepte *Fork* und *Join* in Verbindung mit einem Tokenkonzept realisiert werden. In aPDL bzw. der Basissprache jPDL sind diese als Knoten innerhalb eines Prozessgraphen verwendbar. Dabei wird für jede aus einem *Fork*-Knoten abgehende Transition bei der Abarbeitung ein neuer Kind-Token erstellt. Der ursprüngliche, in den *Fork* eingegangene Eltern-Token, wartet im Fork-Knoten bis alle Kind-Token in einem *Join*-Knoten wieder zusammengekommen sind. Ein *Join*-Knoten vereinigt die parallelen

Prozessabschnitte und gibt nur dann den Kontrollfluss weiter, wenn alle Kind-Token des Eltern-Tokens in diesem Knoten gesammelt sind. Die Kind-Token werden alle beendet und der Eltern-Token wird über die ausgehende Transition des *Join* weiter geleitet.

Innerhalb der Prozessausführung erfolgt eine feinergranulare Modellierung des Systems, bei der die Anzahl der dabei entstehenden Modellierungselemente, wie Komponenten oder Klassen, nicht vorhersehbar ist. Zur Unterstützung der dadurch entstehenden Subprozesse ist eine dynamische Generierung dieser durch die Ausführungsumgebung notwendig. Dies kann durch eine entsprechende Implementierung eines **ActionHandlers** (Kapitel 2.9) erfolgen, wurde aber in dieser Arbeit nicht näher untersucht. Prinzipiell ist es in jPDL möglich durch entsprechenden Java-Code beliebig viele Tokens zu generieren sowie die entsprechenden Ausführungspfade zu starten und zu beenden.

### 5.10 Zusammenfassung

Das Kapitel stellte die Methodik der kontextbasierten Anpassung von Vorgehensmodellen für die Entwicklung laufzeitkritischer Software (kurz: PROKRIS-Methodik) vor. Es wurden die einzelnen Konzepte erläutert und deren Anwendung innerhalb der Methodik dargestellt. Dazu gehören die Modellierung der wiederverwendbaren Prozessbausteine innerhalb der PROKRIS-Prozessbibliothek (NFE-Prozessmuster, NFE-spezifische Prozessbausteine und Ressourcenspezifikation) sowie die Definition der PROKRIS-Facettenklassifikation. Außerdem wurden die drei Schritte Tailoring, Konkretisierung und Individualisierung an Beispielen aus der Erstellung des Vorgehensmodells im Projekt „SuReal“ demonstriert. Abschließend wurde kurz auf die dynamische Ressourcenbindung zur Laufzeit und die Einbindung des Handbuchs in die prozessbasierte Werkzeugumgebung als Konzepte der Ausführung NFE-spezifischer Vorgehensmodelle eingegangen.

Die Vorteile der neuen Methodik liegen einerseits in der Integration der NFE-Prozessmuster in abstrakte NFE-Vorgehensmodelle. Damit wird die systematische und kontinuierliche Unterstützung nicht-funktionaler Laufzeitanforderungen durch Vorgehensmodelle gewährleistet. Auf der anderen Seite stellt die PROKRIS-Methodik erstmals eine Methode zur Vorgehensmodellerstellung bereit, welche sowohl Aspekte der Wiederverwendung von Prozessbausteinen als auch die Wiederverwendung von Vorgehensmodellspezifikationen über die Abstraktionsebenen unterstützt. Das Kapitel verdeutlicht damit die Durchgängigkeit der Vorgehensmodellerstellung vom Makroprozess (abstraktes NFE-Vorgehensmodell), über den Mikroprozess (spezifisches NFE-Vorgehensmodell) bis zur Ausführung.

Abschließend wird der Einfluss der Softwarearchitektur auf die Aufsplittung der NFE-Vorgehensmodelle in einzelne Subprozesse diskutiert. Es werden notwendige Arbeiten sowie die Behandlung der Subprozesse auf den drei Ebenen der PROKRIS-Methodik aufgezeigt.



## 6 Die PROKRIS-Prozessbibliothek

Die PROKRIS-Prozessbibliothek dient als Wissensbasis innerhalb des PROKRIS-Frameworks. In ihr sind die NFE-Prozessmuster, NFE-spezifischen Prozessbausteine und Ressourcen beschrieben. Um die Informationen in der Bibliothek verwalten zu können, werden diese durch die PROKRIS-Facettenklassifikation strukturiert. Zur Modellierung der einzelnen Artefakte innerhalb der Bibliothek wird eine erweiterte Variante des OMG-Standards SPEM 2.0 verwendet. Dieses Kapitel beschreibt nun die definierten Erweiterungen des Metamodells sowie deren Umsetzung in einer Werkzeugunterstützung.

### 6.1 Verwendete Prozessbeschreibungsmittel

Abbildung 6.1 ordnet die einzelnen Komponenten des PROKRIS-Frameworks den verwendeten Beschreibungsmitteln auf Metamodell- und Modellebene zu. Das Prozessmetamodell ist eine Erweiterung des SPEM 2.0 Metamodells. Die Erweiterungen sind notwendig, da die Konzepte des SPEM 2.0 Metamodells nicht die gesamte Methodik des PROKRIS-Frameworks unterstützen. Um die definierten Erweiterungen in das Gesamtbild des SPEM 2.0 Metamodells einzuordnen, zeigt Abbildung 6.2 dessen Paketstruktur. Die einzelnen Pakete stellen folgende Modellierungskonzepte zur Verfügung [Obj08a]:

*Core:* enthält alle Klassen und weitere Verallgemeinerungen, die als Grundlage für alle anderen Pakete dienen.

*Process Structure:* definiert die Basiskonzepte für alle Arten von Prozessmodellen. Dabei liegt der Schwerpunkt auf der statischen Modellierung von Ablaufplänen.

*Process Behavior:* erlaubt die Erweiterung der statischen Ablaufstrukturen um Ausführungsmodelle. SPEM 2.0 definiert keine eigene Ausführungssemantik, sondern nur Schnittstellen für Nutzerimplementierungen. So ist es beispielsweise möglich, UML 2 Aktivitätsdiagramme oder BPEL zu verknüpfen.

*Managed Content:* führt Konzepte ein, um textuelle Beschreibungen zu den Prozesselementen zu verwalten. Diese können entweder allein oder in Verbindung mit der Modellierung einer Prozessstruktur genutzt werden.

*Method Content:* definiert die benötigten Konzepte, um eine Prozesswissensbasis aufzubauen, welche unabhängig von spezifischen Prozessen oder Projekten ist.

*Process With Methods:* definiert Strukturen, um Prozesse (Process Structure) mit Methodeninhalt (Method Content) zu verknüpfen.

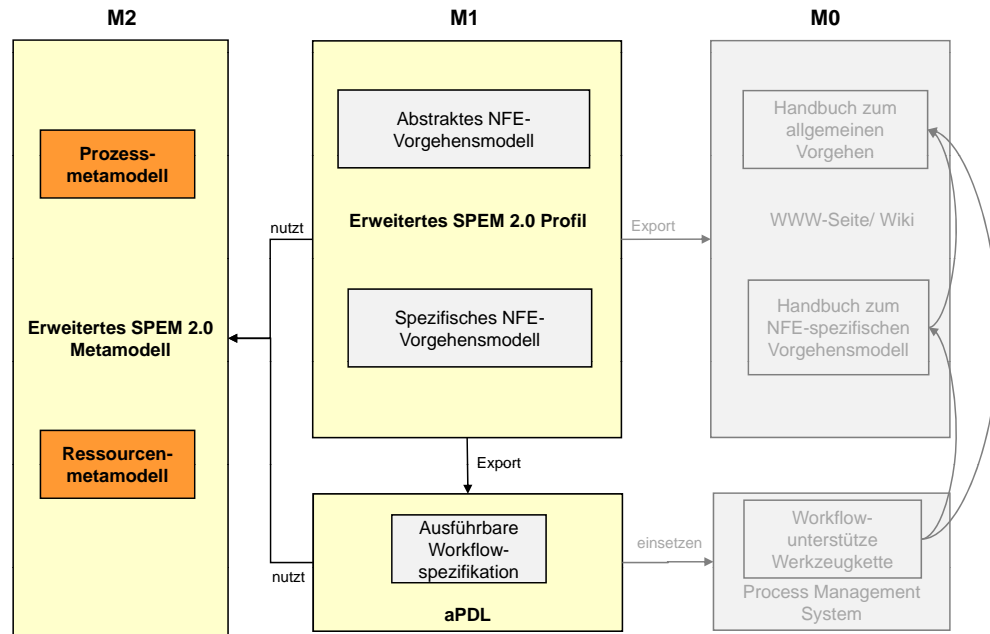


Abbildung 6.1: Eingesetzte Sprachmittel

*Method Plugin:* führt Konzepte für die Modellierung und Verwaltung von großen, skalierbaren, wiederverwendbaren und konfigurierbaren Bibliotheken ein.

Da SPEM 2.0 sowohl ein Metamodell als auch ein zugehöriges Profil definiert, ist es möglich auf der Modellebene (M1) mit diesem Profil zu arbeiten. Dazu sind die notwendigen Erweiterungen im Profil äquivalent zu spezifizieren.

Das Metamodell stellt bewusst keine Ausführungssemantik bereit. Um die Anbindung an verschiedene Ausführungssprachen zu ermöglichen, sind sowohl Schnittstellen von Elementen der Methodenbibliothek als auch Elementen der Prozessstruktur definiert. Über diese kann die Abbildung der Metamodellkonzepte auf Ausführungssprache aPDL realisiert werden (Abbildung 6.1, unten).

## 6.2 Erweiterungen des Prozessmetamodells SPEM 2.0

Obwohl sich das Prozessmetamodell SPEM 2.0 bei der Anwendung der Methodik und ihrer Werkzeuge als ausgereift und mächtig herausgestellt hat, waren in einigen für die PROKRIS-Methodik wesentlichen Bereichen Defizite vorhanden. Daher wurden in folgenden Bereichen Erweiterungen des Metamodells definiert:

- Modellierung von NFE-Prozessmustern und deren Verwendung im abstrakten NFE-Vorgehensmodell
- Modellierung von NFE-spezifischen Prozessbausteinen und deren Verwendung im spezifischen NFE-Vorgehensmodell

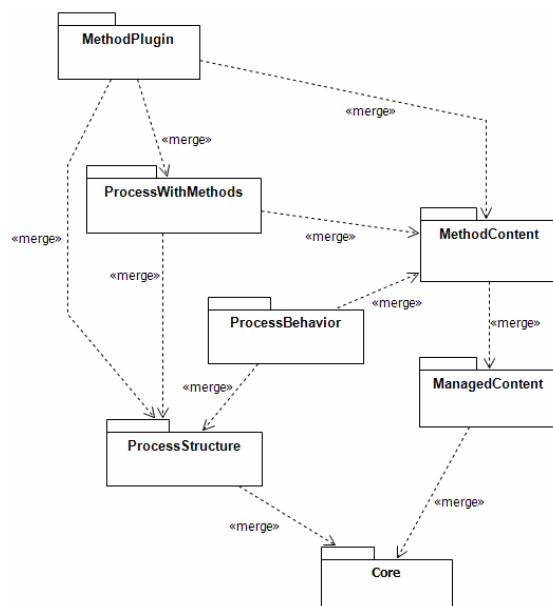


Abbildung 6.2: Struktur des SPEM 2.0 Metamodells

- Unterscheidung der Abstraktionsebenen der NFE-Vorgehensmodelle
- Modellierung von abstrakten und konkreten Ressourcen
- Anbindung einer Ausführungssprache
- Verwendung der Ressourcen in statischen und ausführbaren Prozessen
- Modellierung der Kontexte und deren Verbindung zu den NFE-Prozessmustern bzw. -bausteinen

Konsistent dazu wurde das ebenfalls in SPEM 2.0 definierte UML-Profil erweitert. Als Besonderheit ist hier anzumerken, dass die statische Semantik des Profils durch das Metamodell selbst bereitgestellt wird ([Obj08a], S.19).

Die definierten Erweiterungen und Änderungen sind in den folgenden Abbildungen grau hinterlegt.

### 6.2.1 NFE-Prozessmuster und -bausteine

Der Inhalt der NFE-Prozessmuster und -bausteine soll für den Prozessingenieur sichtbar sein, da er die notwendigen Methoden innerhalb der Komponente durch konkrete Techniken ersetzen soll. Dazu ist es notwendig, Teilprozesse mit mehreren Aktivitäten und Artefakten gekapselt als „White Box“-Komponente beschreiben zu können.

SPEM 2.0 bietet bereits ein Metamodellelement **ProcessComponent** an, welches jedoch nur eine einzelne Aktivität als Black-Box-Komponente kapselt. Als Schnittstelle werden

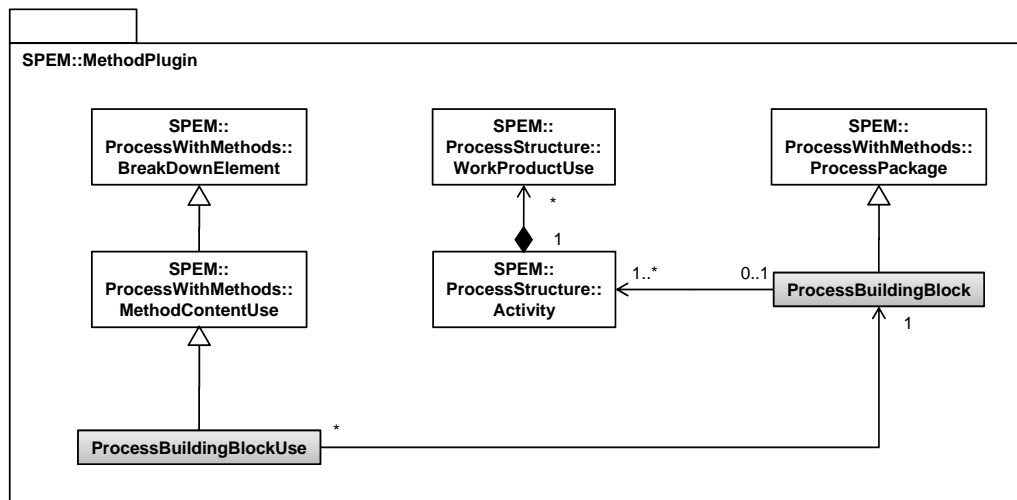


Abbildung 6.3: Metamodell der PROKRIS-Prozessartefakte

die Artefakte des Elements **Activity** verwendet. Die Umsetzung der Aktivität ist dabei nicht relevant. Daher ist dieses Metamodellkonstrukt für den PROKRIS-Ansatz nicht geeignet. Weiterhin spezifiziert der Standard im SPEM 2.0 Profil ein **ProcessPattern**. Dies erfolgt innerhalb eines vordefinierten Methoden-Plugins (**Base Plugin**), welches eine Instanz des Metamodellpakets **SPEM::MethodPlugin** darstellt.

Das **ProcessPattern** beschreibt innerhalb eines generellen Prozesses einen speziellen Prozess, der eine wiederverwendbare Gruppe von Aktivitäten für einen konsistenten Entwicklungsansatz allgemeiner Problemstellungen bereitstellt.

Da diese Definition eine White-Box-Sicht auf einen Teilprozess modelliert, ist sie prinzipiell auf das NFE-Prozessmuster bzw. den NFE-Prozessbaustein anwendbar. Allerdings existiert in SPEM 2.0 dafür kein entsprechendes Metamodellelement und somit keine semantisch eindeutige Definition. Um die Konsistenz mit den darauf aufbauenden Werkzeugen weiterhin zu gewährleisten, soll das bestehende Metamodell nur durch Erweiterungen verändert werden. Daher wird nicht das fehlende Metamodellelement für **ProcessPattern**, sondern sowohl im Metamodell als auch im Profil das neue Element **ProcessBuildingBlock** eingeführt.

Abbildung 6.3 zeigt das Metamodellelement und dessen Einordnung in die Paketstruktur. Die Erweiterung erfolgt im Paket **MethodPlugin**. Ein **ProcessBuildingBlock** ist ein spezieller **Process**, welcher den Prinzipien der Kapselung genügt. Er enthält einen (Teil-)Prozess, welcher aus mehreren Aktivitäten bestehen kann. Die Aktivitäten arbeiten mit einer Anzahl von Arbeitsprodukten, deren Relation über **WorkProductUse** definiert ist. Die korrekte Verwendung eines Prozessmusters bzw. -bausteins innerhalb eines Prozesses wird über das neue Element **ProcessBuildingBlockUse** gewährleistet.

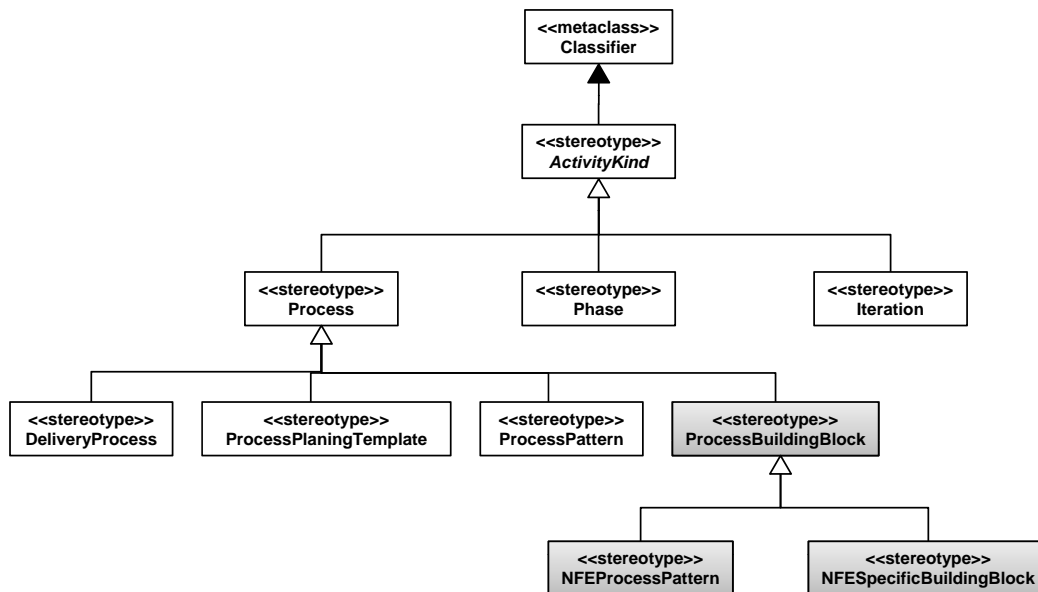


Abbildung 6.4: Klassifikation der Prozessartefakte im Profil

Die Einführung eines entsprechenden Stereotyps ist in Abbildung 6.4 dargestellt. Es werden außerdem zwei Unterklassen definiert: `NFEPattern` und `NFESpecificBuildingBlock`. Diese entsprechen der Unterscheidung in NFE-Prozessmuster und NFE-spezifische Prozessbausteine und ermöglichen dadurch deren Differenzierung innerhalb der Prozessmodellierung.

### 6.2.2 Ressourcen - Modellierung der Organisationsstruktur

Die Ressourcenspezifikation ermöglicht die Beschreibung konkreter Ressourcen einer Organisation. SPEM 2.0 definiert zu einer Aufgabe (`Task`) zugehörige Rollen (`Role`) und Werkzeuge (`Tool`). Dabei handelt es sich jedoch nicht um konkrete Ressourcen, sondern abstrakte Beschreibungen der notwendigen Eigenschaften. Die Modellierung der konkreten Ressourcen ist nicht Bestandteil der SPEM-Spezifikation. Daher muss das Metamodell um die notwendigen Konzepte erweitert werden. Abbildung 6.5 zeigt die entsprechenden Metamodellkonstrukte und deren Einbindung in den Standard. Die Konzepte werden in das Paket `MethodContent` integriert, da dort die Basiselemente des Methodeninhalts der Bibliothek enthalten sind.

Definiert wird die Metaklasse `Resource`, welche von `MethodContentElement` abgeleitet ist. Als Unterklassen werden drei Arten von Ressourcen unterschieden: Personen (`Person`), Werkzeuge (`Program`) und Daten (`Data`). Die Definition der *Personen* ist relativ einfach gehalten. Sie besitzen einen Namen und einen Status über den ihre Verfügbarkeit verwaltet wird. Login und Passwort sind zur Vereinfachung innerhalb der selben Klasse modelliert, was natürlich aus Datenschutzgründen getrennt geschehen sollte. *Daten* repräsentieren die reinen Arbeitsdaten innerhalb des Prozesses und keine Daten

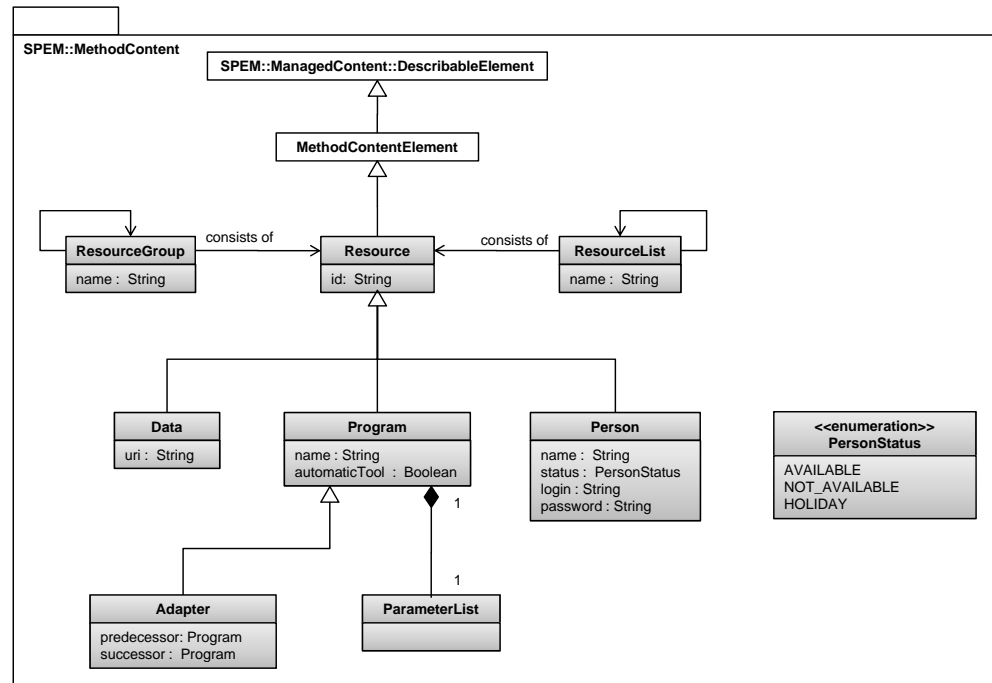


Abbildung 6.5: Metamodell der Ressourcenspezifikation

des Kontrollflusses (z.B. Statusinformationen). *Werkzeuge* werden zur Abgrenzung der abstrakten Werkzeugbeschreibung unter *Tool* als **Program** bezeichnet, da es sich im Falle der bereitzustellenden automatischen Werkzeugketten um Aneinanderreihungen von Programmen handelt.

Es existiert außerdem eine spezielle Art von Werkzeugen, der Adapter (**Adapter**). Adapter beschreiben Werkzeuge, die zur Transformation von Ausgabedaten eines Werkzeuges in Eingabedaten eines anderen benötigt werden. Dazu zählen beispielsweise Transformationswerkzeuge. Ein Adapter spezifiziert daher ein Vorgängerwerkzeug, dessen Ausgabedaten auf die Eingabedaten des Nachfolgewerkzeuges abgebildet werden.

Programme besitzen eine Parameterliste. Diese beinhaltet die für eine Integration in eine Werkzeugumgebung notwendigen werkzeugspezifischen Informationen. Zur Systematisierung können allgemeine Typen von Werkzeugen differenziert werden. Die Klassifikation der Werkzeuge und deren verschiedene Ausprägungen der Parameterlisten sind in Abbildung 6.6 zu sehen. So benötigt ein Kommandozeilenwerkzeug die Spezifikation von Ausführungspfad und -datei zum Aufruf des Werkzeugs innerhalb einer Werkzeugkette. Ein Eclipse Plugin dagegen wird über die PluginID und weitere Informationen, wie die Klasse, in der sich die aufzurufende Methode befindet (**methodClass**), den Namen der Methode (**method**) und deren Parameter (**parameterClass**), aufgerufen. Zusätzlich wurde ein Ant-Werkzeug und zwei weitere Eclipse-spezifische Werkzeuge, Editor und LaunchShortcut definiert. Letzteres ermöglicht das Einbinden von Ausführungskonfigurationen innerhalb von Eclipse, beispielsweise den Vorgang des Kom-

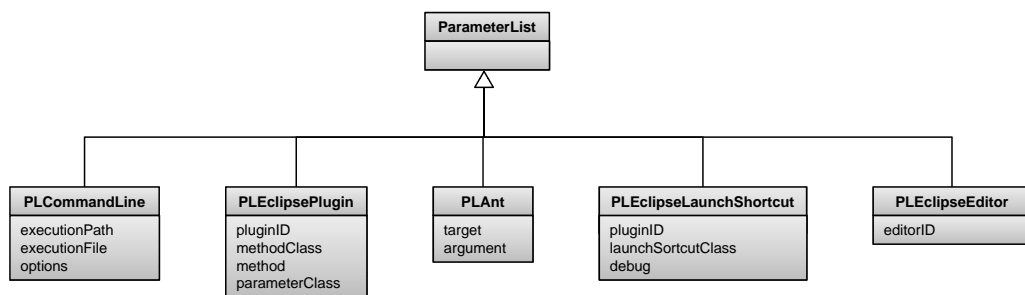


Abbildung 6.6: Klassifikation der Werkzeuge (PL: Parameterliste)

Stereotyp	Symbol	Stereotyp	Symbol
Resource		ParameterList	
ResourceGroup		PLCommandLine	
ResourceList		PLEclipsePlugin	
Program		PLAnt	
Data		PLEclipseLaunchShortCut	
Person		PLEclipseEditor	

Abbildung 6.7: Definierte Symbole der Stereotypen von Ressourcen

pilierens und anschließend dem Ausführen einer java-Datei mit main-Methode. Die Klassifikation der unterschiedlichen Werkzeuge ist nach Bedarf erweiterbar.

Konkrete Ressourcen können in zwei verschiedenen Formen gruppiert werden (Abbildung 6.5). Einerseits ist es möglich, alle verfügbaren konkreten Ressourcen in einer getypten Liste (**ResourceList**) anzuordnen. Andererseits können die Ressourcen zur Unterstützung der Konzepte der Ressourcenbindung in hierarchischer Form gruppiert werden (**ResourceGroup**).

Die Erweiterungen im Metamodell wurden entsprechend vorgenommen. Abbildung 6.7 gibt einen Überblick über die definierten Symbole der Stereotypen.

Die abstrakten Ressourcenbeschreibungen sind in die statische Prozessstruktur einzubinden. Die konkreten Ressourcen werden an die ausführbare Beschreibung gebunden. Dies wird für die Metamodellebene in den nächsten beiden Absätzen beschrieben.

### Ressourcen in der statischen Prozessbeschreibung

Innerhalb der Prozessmodellierung werden unter Nutzung der Pakete **ProcessStructure** und **ProcessWithMethods** die Aufgaben und damit die abstrakten Rollen und Werkzeugbeschreibungen an Aktivitäten gebunden und somit in eine statische Ablaufstruktur eines Prozesses integriert. Abbildung 6.8 zeigt im oberen Teil die Einbindung einer Aktivität in eine Ablaufstruktur über das Element **BreakdownElement** im Paket **ProcessStructure**. Dieses wird im Paket **ProcessWithMethods** so erweitert, dass Aufgaben-, Arbeitsprodukt- und Rollenbeschreibungen über **RoleUse**, **TaskUse** und **WorkProductUse** zugeordnet werden können. Das PROKRIS-Framework benötigt

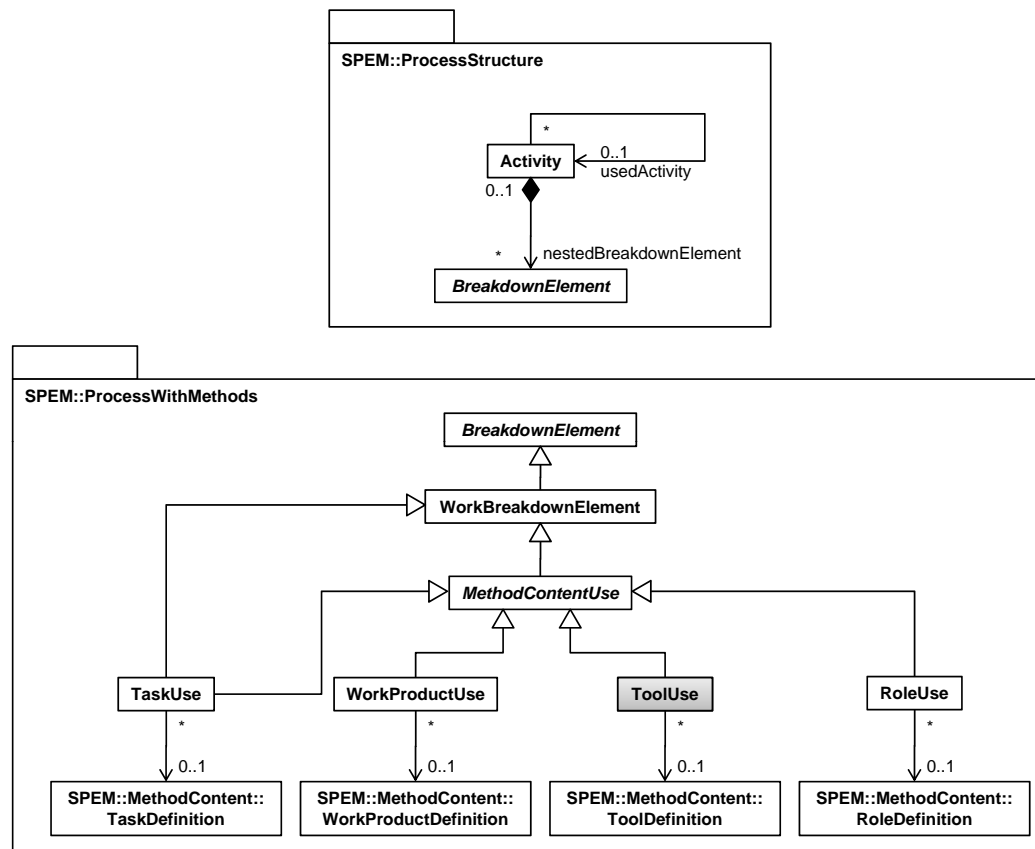


Abbildung 6.8: Einbindung abstrakter Ressourcen in die statische Prozessstruktur



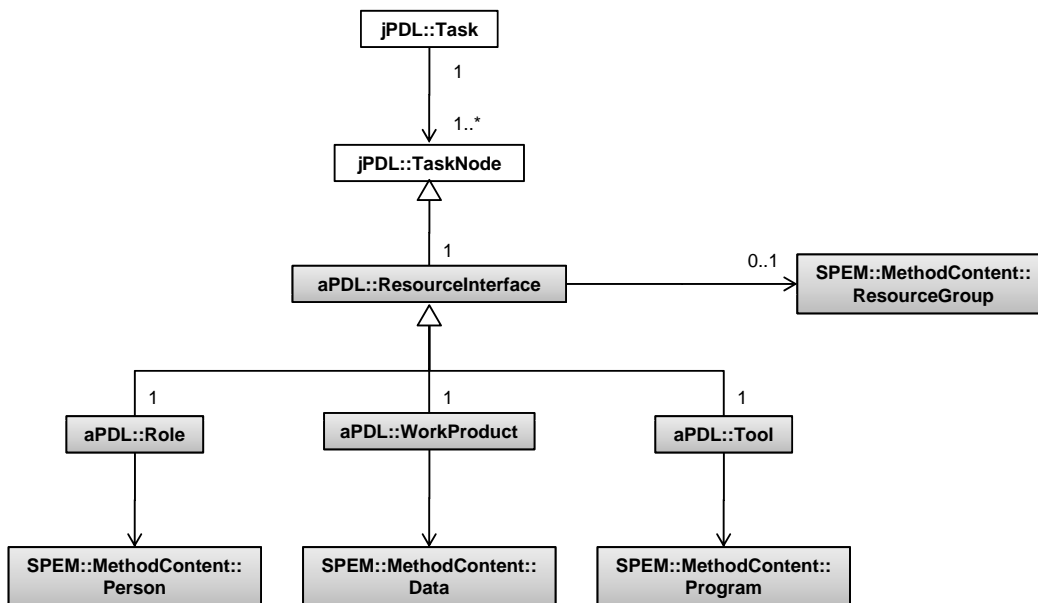


Abbildung 6.9: Bindung konkreter Ressourcen an Ressourcenschnittstellen der ausführbaren Prozessbeschreibung

zusätzlich eine Verknüpfung zu abstrakten Werkzeugbeschreibungen. Die Beschreibung an sich ist im Standard unter `SPEM::MethodContent::ToolDefinition` bereits definiert. Die Einbindung in einen Prozess ist nur im Profil über den Stereotyp `ToolMentor` als Hilfe (Guidance) möglich. Daher wird in Anlehnung an die Abstraktion der Relationen zwischen den anderen abstrakten Ressourcen und dem Prozess das Metamodellelement `ToolUse` eingeführt (Abbildung 6.8 unten).

### Ressourcen in der ausführbaren Prozessbeschreibung

aPDL erweitert jPDL um notwendige Konzepte zur Unterstützung der dynamischen Ressourcenbindung. Abbildung 6.9 zeigt die entsprechenden Elemente. Hauptelement von aPDL sind `TaskNodes`, welche wiederum ein oder mehrere Aufgaben (`Task`) enthalten. Diese sind von jPDL übernommen. Zur Bindung der Ressourcen wurde eine spezielle Schnittstelle `ResourceInterface` am Element `Task` eingeführt. Diese Schnittstelle besitzt drei Ausprägungen: `Role`, `WorkProdukt` und `Tool`. Diese werden bei der Übersetzung von statischen in ausführbare aPDL-Prozesse erzeugt. Einen Auszug der definierten Abbildungen von Elementen des SPEM 2.0 Metamodells auf aPDL-Elemente listet Tabelle 6.1 auf. An die definierten Ressourcenschnittstellen können entsprechende konkrete Ressourcen gebunden werden, d.h. an `Role` Elemente von `Person`, an `WorkProduct` Elemente von `Data` und an `Tool` Elemente von `Program`. Außerdem können Ressourcengruppen in die Prozessbeschreibung integriert werden.

SPEM Element bzw. definierte Erweiterung	aPDL Element
Delivery Process	Process Definition
Capability Pattern, Activity, Phase, Iteration	Super State
Task	Task Node
Step	Task
Predecessor	Transition
WorkProductUse	WorkProduct
RoleUse	Role
ToolUse	Tool

Tabelle 6.1: Abbildung von SPEM (erweitert) auf aPDL-Elemente

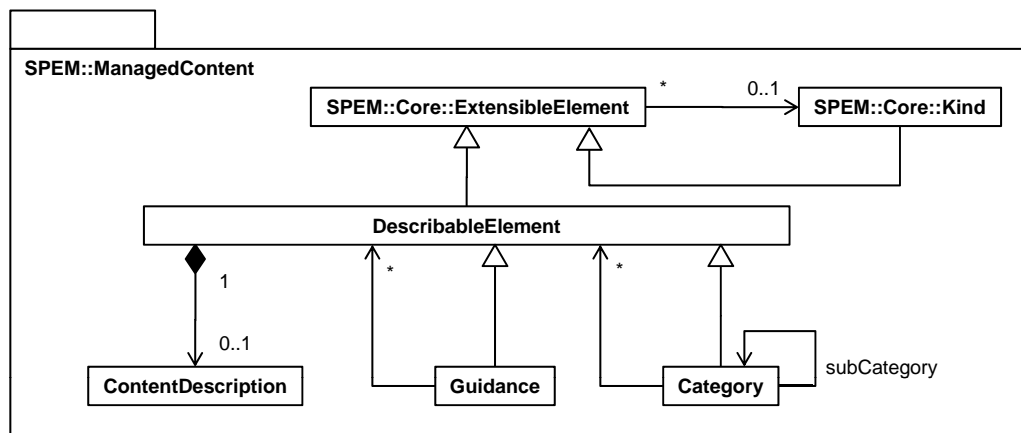


Abbildung 6.10: Modellierung der Kontexte

### 6.2.3 Kontexte und Facetten

Um die vielfältige Gesamtmenge der Elemente der PROKRIS-Prozessbibliothek überschaubar und damit besser nutzbar zu machen, ist deren Strukturierung zwingend erforderlich. Dies bedeutet, dass eine geeignete Möglichkeit für die effiziente Verwaltung und Nutzung der Prozessbausteine zu finden ist. PROKRIS-Methodik führt dazu die Konzepte Kontext und Facette ein, deren Umsetzung im Metamodell im Folgenden diskutiert wird.

SPEM 2.0 schlägt von Haus aus eine Plugin-Architektur vor, um die Wiederverwendbarkeit einzelner Prozessbeschreibungen zu ermöglichen. So kann die Bibliothek in verschiedene Methodenbibliotheken, z.B. eine für RUP und eine für MDA, aufgeteilt werden. Diese können sowohl einzeln als auch zusammen für die Modellierung von Prozessen eingesetzt werden. Außerdem können die Elemente des Methodeninhalts Standardkategorien zugeordnet werden, die die Elemente ausgehend von ihrem Typ sortieren. Das Basis-Methoden-Plugin von SPEM 2.0 definiert, dass sich Rollen in **Role Sets** zu-

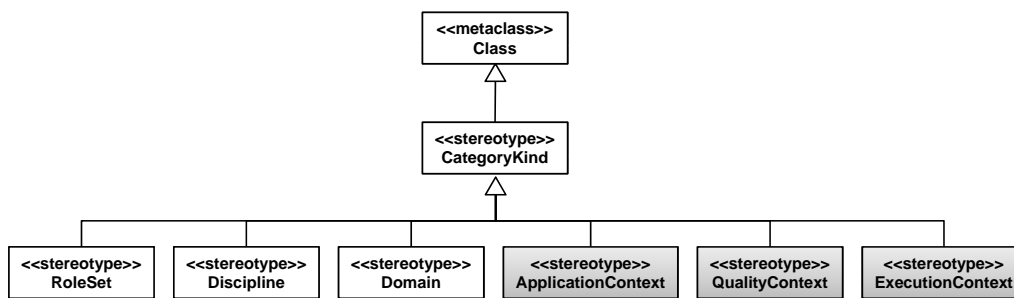


Abbildung 6.11: Stereotypen von Kontexten

sammenfassen lassen sowie Aufgaben in **Disciplines** und **Workproducts** in **Domains** eingeordnet werden. Über diese drei definierten Standardkategorien hinaus existiert die Möglichkeit, eigene Kategorien anzulegen, was für die Unterstützung der PROKRIS-Methodik notwendig ist. In dieser werden die verfügbaren NFE-Prozessbausteine anhand verschiedener Kontexte in die PROKRIS-Facettenklassifikation eingordnet. Daher sind diese als Kategorien abzubilden.

Die Standardkategorien werden in SPEM 2.0 durch das Metamodellelement **Category** definiert, welches auch zur Abbildung der PROKRIS-Klassifikation verwendet wird. Abbildung 6.10 zeigt das Modell. Die PROKRIS-Methodik unterscheidet zusätzlich die drei Kontexte: Anwendungs-, NFE- und Ausführungskontext. Zu deren Unterscheidung werden im erweiterten Profil folgende Stereotypen eingeführt: **ApplicationContext**, **QualityContext** und **ExecutionContext** (Abbildung 6.11).

Den Kontexten sind Facetten zugeordnet, welche eine Facettenklassifikation aufbauen. Diese Facetten können durch das Metamodellelement **subCategory** abgebildet werden. Durch dieses Element können Hierarchien von Kategorien aufgebaut werden. Der Standard definiert weiterhin, dass diese Beziehung keine strikte Verschachtlung beschreibt, sondern, dass eine **Category** eine **subCategory** anderer **Categories** sein kann. Dies entspricht der Definition einer Facettenklassifikation [Pri08], wie sie für die PROKRIS-Bibliothek benötigt wird. Es sind daher keine weiteren Erweiterungen auf der Metamodellebene notwendig.

## 6.3 Umsetzung des Metamodells in Werkzeugunterstützung

Zur Anwendung des erweiterten Metamodells wird eine entsprechende Werkzeugunterstützung benötigt. Wie bereits in Abbildung 6.1 dargestellt, können die Erweiterungen des SPEM 2.0 Metamodells in allgemeine Prozessmetamodellkonzepte und dem Ressourcenmetamodell unterschieden werden. Die Abbildung zeigt außerdem, dass es auf Modellebene eine Trennung der Modellierungskonzepte der statischen Prozessbeschreibung mittels des erweiterten SPEM 2.0 Profils und der ausführbaren Beschreibung mit aPDL existiert. Diese Trennung spiegelt sich auch in den Modellierungswerkzeugen wieder. Während das Prozessmetamodell der Beschreibung der statischen NFE-

Vorgehensmodelle dient, wird das Ressourcenmetamodell vorwiegend während der Modellierung und Individualisierung der ausführbaren Modelle eingesetzt. Da sich beide Arten von Vorgehensmodellen auch in der verwendeten Modellierungssprache unterscheiden, wurden verschiedene Wege zur Umsetzung in einem Modellierungswerkzeug gewählt.

### Prozessmetamodell

Die Modellierung der NFE-Prozessmuster, der NFE-spezifischen Umsetzungen und der statischen Vorgehensmodelle erfolgt unter Nutzung einer Erweiterung des EPF Composers innerhalb des PROKRIS-Frameworks. Da der EPF Composer eine Referenzimplementierung des SPEM 2.0 Profils ist, können die Erweiterungen des Metamodells unter Nutzung des Profils bei der Modellierung verwendet werden. Kapitel 5 zeigte ausgewählte Beispiele.

### Ressourcenmetamodell

Wie bereits beschrieben, erfolgt die Bindung der konkreten, in der Ressourcenspezifikation modellierten Ressourcen an die ausführbare Prozessbeschreibung über Ressourcenschnittstellen innerhalb der Sprache aPDL. Da diese im XML-Format spezifiziert wird, ist es naheliegend, auch die Ressourcenspezifikation innerhalb des PROKRIS-Frameworks als XML-Datenstruktur zu verwalten. Die XML-Schema Definition setzt demnach die Konstrukte zur Ressourcenmodellierung der Metaebene M2 um. Auf Basis des XML-Schemas wurde ein Ressourceneditor implementiert. Dieser erzeugt eine XML-basierte Ressourcenspezifikation. Listing 6.1 stellt auszugsweise die Spezifikation unter Nutzung des XML-Schemas anhand eines Beispiels vor.

```

1 <resourceGroup>
2   <category name="Requirements">
3     <resourceGroup name="RequirementsTeam" ref="aa1 ff1" />
4     <resourceGroup name="RequirementsData" ref="ucm" />
5     <resourceGroup name="RequirementsTool" ref="Word Notepad" />
6   </category>
7   ...
8 </resourceGroup>
9 ...
10 <resourceList>
11 <resource id="Notepad" name="Notepad" type="Program">
12   <tool:Tool>
13     <tool:Name>CommandLine-Tool Notepad</tool:Name>
14     <tool:Description> simple tool for documentation </tool:Description>
15     <tool:ParamList xsi:type="tool:PLCommandLine">
16       <tool:ExecutionPath>C:\Windows\</tool:ExecutionPath>
17       <tool:ExecutionFile>notepad.exe</tool:ExecutionFile>
18       <tool:Options>[INPUT]</tool:Options>
19     </tool:ParamList>
20     <tool:OutputURL>[CURRENTINPUTURL]</tool:OutputURL>
21     <tool:AutomaticTool>false</tool:AutomaticTool>
22   </tool:Tool>
23 </resource>
24 <resource id="Word" name="Word" type="Program">
25   <tool:Tool>
26     <tool:Name>CommandLine-Tool Word</tool:Name>
27     <tool:Description> complex tool for documentation </tool:Description>
28     <tool:ParamList xsi:type="tool:PLCommandLine">
29       <tool:ExecutionPath>C:\Programme\MicrosoftOffice2007Pro\</tool:ExecutionPath>

```

```

31  <tool:ExecutionFile>winword.exe</tool:ExecutionFile>
    <tool:Options>[INPUT]</tool:Options>
  </tool:ParamList>
33  <tool:OutputURL>[CURRENTINPUT.URL]</tool:OutputURL>
    <tool:AutomaticTool>false</tool:AutomaticTool>
35  </tool:Tool>
  </resource>
37  ...
  <resource type="Data" id="ucm" name="UseCaseModelle"
    url="[WORKSPACE]\Anforderungen.txt"/>
39  ...
  <resource type="Person" id="aal" name="Albert Alleskoenner" login="aal"
    password="1A+b" status="available"/>
41  <resource type="Person" id="ff1" name="Felix Fuchs" login="ff1" password="lf3*"
    status="available"/>
  ...
43 </resourceList>

```

Listing 6.1: Auszug aus einer Ressourcenspezifikation

Das Beispiel zeigt die Definition zweier Kommandozeilenwerkzeuge, zweier Personen und einer Datendefinition innerhalb des Bereichs der Ressourcenliste (ab Zeile 10). In Zeile 1-8 werden diese zu Gruppen zusammengefasst. Diese Möglichkeit der hierarchischen Gruppierung wird zur Klassifikation der Ressourcen bezüglich des Ausführungskontexts genutzt. Es wird eine Obergruppe **Requirements** eingeführt, also nach der Entwicklungsphase gruppiert. In der zweiten Hierarchieebene erfolgt die Einteilung nach dem Typ der Ressource: Personen, Daten, Werkzeuge. Dabei enthalten **RequirementsTeam** und **RequirementsTool** jeweils zwei Referenzen auf konkrete Ressourcen. Dies ermöglicht es, die Entscheidung über den Einsatz der jeweiligen Personen bzw. hier auch der Werkzeuge (Word bzw. Notepad) zur Laufzeit, während der dynamischen Ressourcenbindung innerhalb der prozessbasierten Ausführungsumgebung, zu fällen.

Anhang C enthält eine ausführlichere Beispielspezifikation des Szenarios für die Evaluierung mittels der „SuReal“-Fallstudie.

Das Modell des XML-Schemas besteht aus zwei Teilen. Einer Definition der Konzepte der Werkzeugbeschreibungen und der eigentlichen Ressourcenspezifikation. Die Spezifikation der Integration der Werkzeuge wird von der Architektur der Ausführungsumgebung (hier der Workflowmaschine jBPM) beeinflusst. Durch die separate Beschreibung ist dieser Teil der Spezifikation flexibel austauschbar. Listing 6.2 zeigt einen entsprechenden Auszug aus der Schema-Definition. In Zeile 16 und 22 wird auf die separate XML-Schema-Definition der Werkzeuge (tool.xsd) über `tool:Tool` verwiesen.

```

1  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tool="http://www....">
  <xs:import namespace="http://www.example.org/Tool" schemaLocation="tool.xsd" />
3  ...
  <xs:complexType abstract="true" name="Resource">
5    <xs:attribute name="id" type="xs:string"/>
  </xs:complexType>
7
  <xs:complexType abstract="true" name="Program">
9    <xs:complexContent>
      <xs:extension base="resource:Resource"/>
11    </xs:complexContent>
  </xs:complexType>

```

```
13 <xs:complexType implements="Program" name="ExternalTool">
15   <xs:complexContent>
17     <xs:extension base="tool:Tool" />
18   </xs:complexContent>
19 </xs:complexType>
20
21 <xs:complexType implements="Program" name="Adapter">
22   <xs:complexContent>
23     <xs:extension base="tool:Tool" />
24     <xs:attribute name="predecessor" type="tool:Tool" />
25     <xs:attribute name="successor" type="tool:Tool" />
26   </xs:complexContent>
27 </xs:complexType>
...
</xs:schema>
```

Listing 6.2: Auszug aus dem XML-Schema zur Ressourcenspezifikation

## 6.4 Zusammenfassung

Dieses Kapitel beschrieb die zur Unterstützung der PROKRIS-Methodik notwendigen Erweiterungen des SPEM 2.0 Metamodells. Konsistent dazu wurde das ebenfalls im Standard definierte Profil erweitert. Abschließend wird auf die Umsetzung des erweiterten Metamodells innerhalb einer adäquaten Werkzeugunterstützung zur Umsetzung der PROKRIS-Prozessbibliothek eingegangen. Diese ist in zwei Werkzeuge aufgeteilt. Ein erstes zur Beschreibung der wiederverwendbaren NFE-Prozessmuster deren Instanzen und zugehörigen Teilspezifikationen auf Basis des EPF Composers. Der Ressourceneditor als zweites Werkzeug dient der Modellierung und Klassifikation der Ressourcen.

## 7 Der PROKRIS-Prozesskomposer

Der PROKRIS-Komposer dient innerhalb des PROKRIS-Frameworks zur Komposition und Anpassung der erweiterten Vorgehensmodelle zur Entwicklung laufzeitkritischer Software. Das folgende Kapitel definiert die dazu notwendigen Modellierungskonzepte innerhalb des Prozess-Metamodells. Darauf aufbauend werden wichtige Aspekte der Architektur und der Umsetzung behandelt. Außerdem wird auf die Übersetzung der statischen in ausführbare Vorgehensmodelle, speziell die Unterstützung der Integration des Handbuchs eingegangen.

### 7.1 Erweiterte Metamodellkonzepte zur Komposition

Zur Unterstützung der Komposition und Anpassung der erweiterten Vorgehensmodelle innerhalb der PROKRIS-Methodik werden folgende Konzepte auf der Metamodellebene benötigt:

- Unterscheidung der Abstraktionsebenen der NFE-Vorgehensmodelle
- Modellierung der Integration von Prozessmustern im abstrakten NFE-Vorgehensmodell
- Modellierung der Integration von Prozessbausteinen im spezifischen NFE-Vorgehensmodell
- Modellierung der ausführbaren Vorgehensmodelle
- Modellierung der Verwendung der Ressourcen in statischen und ausführbaren Vorgehensmodellen
- Integration des Kompositionsfilters

Bis auf den letzten Punkt wurden die Erweiterungen bereits im Kapitel 6 beschrieben. Daher wird im Folgenden nur auf das Metamodell des Kompositionsfilters eingegangen.

#### Kompositionsfilter als Erweiterung von SPEM 2.0

Der Kompositionsfilter wird als Erweiterung einer Konfiguration (`MethodConfiguration`) eingeführt. Abbildung 7.1 zeigt das entsprechende Metamodell. Eine Konfiguration definiert eine Untermenge einer Methodenbibliothek (`MethodLibrary`) und kann somit als Filter dienen. Der Kompositionsfilter (`FacetbasedCompositionFilter`) erweitert diesen Ansatz um die Suche innerhalb der PROKRIS-Facettenklassifikation.

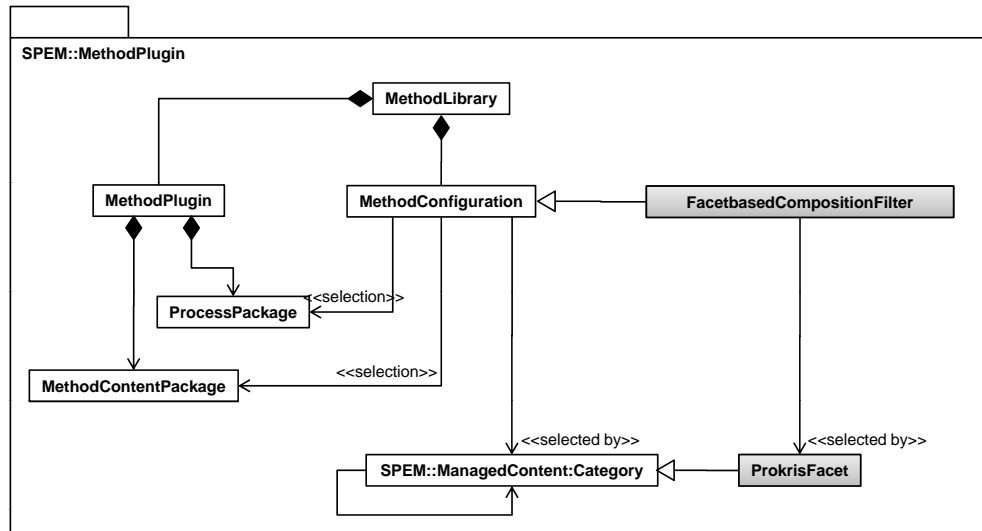


Abbildung 7.1: Umsetzung des Kompositionsfilters als Erweiterung von SPEM 2.0

Wichtig ist dabei, dass die PROKRIS-Facetten für die gesamte PROKRIS-Bibliothek definiert werden. Damit wird sichergestellt, dass die PROKRIS-Facetten übergreifend über verschiedene **MethodPlugins** gleich sind. Diese Bedingung ist wichtig, da der Standard eine weitere Möglichkeit als Definition sogenannter **Custom Categories** vorsieht, welche aber in jedem **MethodPlugin** verschieden sein können.

## 7.2 Tailoring und Konkretisierung der statischen Modelle

Auf die Nutzung des Kompositionsfilters innerhalb der Anpassungsschritte Tailoring und Konkretisierung wurde bereits bei der Beschreibung der Anwendung der PROKRIS-Methodik eingegangen. Im Folgenden wird nun auf die Umsetzung innerhalb der Architektur des PROKRIS-Frameworks eingegangen.

### Werkzeugunterstützung

Zur Unterstützung und Machbarkeitsanalyse der Komposition der statischen Vorgehensmodelle wurde eine Erweiterung für den EPF Composer entwickelt. Dieser gilt als Referenzimplementierung des Metamodells SPEM 2.0 und stellt daher bereits die Grundfunktionalität zur Komposition von Prozessen bereit. Zur Unterstützung der PROKRIS-Methodik musste folgende Funktionalität ergänzt werden:

- Unterstützung des Kompositionsfilters, d.h. Filterung der wiederverwendbaren Prozessbausteine anhand der Zuordnung zur PROKRIS-Facettenklassifikation
- Ersetzen von Elementen in der Ablaufstruktur (**Work Breakdown Structure**) des Prozesseditors unter Einsatz des Kompositionsfilters



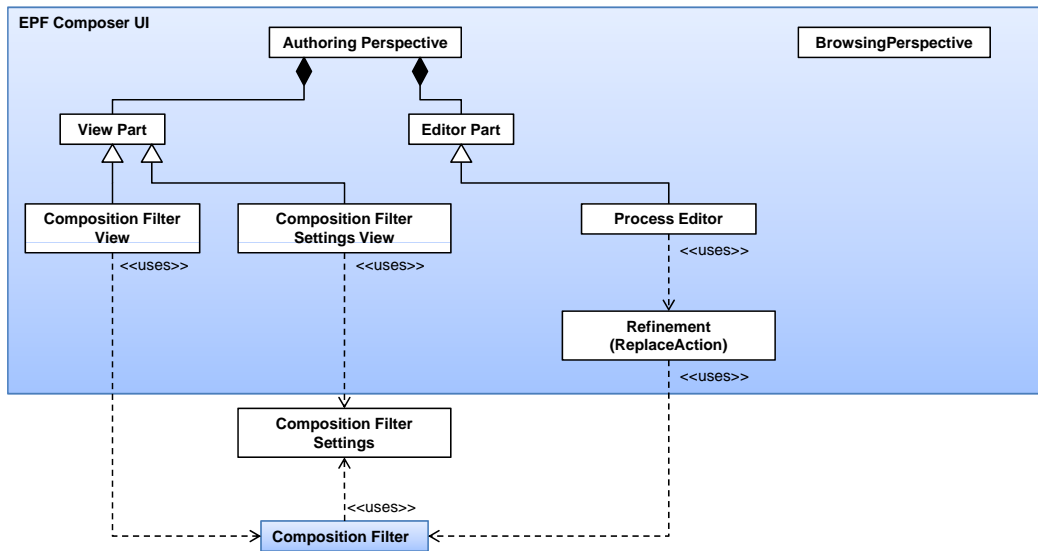


Abbildung 7.2: Erweiterung der Architektur des EPF Composers um den Kompositionsfilter

Die Architektur des Kompositionsfilters folgt der *Model-View-Controller*-Architektur. Das Modell ist die Datenstruktur, welche durch den EPF Composer aufgebaut wird. Neu hinzugefügt wird der Filtermechanismus als *Controller* und die Benutzeroberfläche für die Einstellung des Filters sowie die Ausgabe der relevanten Prozessbausteine als *View*.

Abbildung 7.2 gibt einen Überblick über die Erweiterung der Architektur des EPF Composers im Bereich der Nutzerschnittstelle (EPF Composer UI). Es wurde darauf geachtet, dass der Filtermechanismus in verschiedene Funktionen der Nutzerschnittstelle integriert werden kann. So ist der Kompositionsfilter eigenständig als Dialogfenster (Composition Filter View) verfügbar, wird aber ebenso in den entsprechenden Dialogen zur NFE-spezifischen Konkretisierung des Vorgehensmodells innerhalb des Prozesseditors verwendet (Refinement (Replace Action)). Die erste Variante wird während des Anpassungsschritts *Tailoring*, also zum Suchen und Integrieren geeigneter NFE-Prozessmuster in das abstrakte NFE-Vorgehensmodell eingesetzt. Auszüge aus den Dialogen und damit deren Benutzeroberflächen wurden bei der Beschreibung der Methodik im Kapitel 5 gezeigt.

Weitere detailliertere Erklärungen zur Implementierung können [RSS09] entnommen werden.

### Konfigurationen des Kompositionsfilters

Die Umsetzung des Kompositionsfilters unterscheidet zwei unterschiedliche Möglichkeiten der Auswahl von wiederverwendbaren Prozessbausteinen anhand der PROKRIS-Facetten, die *AND*-Verknüpfung und die *OR*-Verknüpfung.

Im *AND*-Modus werden nur die untersten Elemente, also die Blattknoten, der Facettenklassifikation ausgewählt. Die Ergebnisse einer Anfrage des Kompositionsfilters

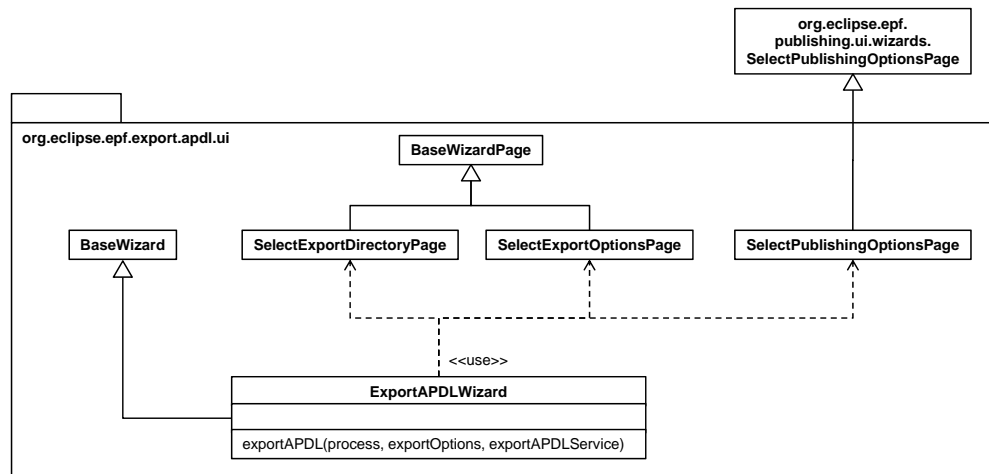


Abbildung 7.3: Auszug aus der Architektur der aPDL-Export-Schnittstelle

beinhalten dann nur NFE-Prozessbausteine, die alle ausgewählten Kriterien erfüllen.

Im *OR*-Modus können auch Kategorien der Facettenklassifikation ausgewählt werden, die selbst noch weitere Unterkategorien enthalten. Der Kompositionsfilter arbeitet dann bis zum Knoten des ausgewählten Elements im *AND*-Modus. Für alle Unterkategorien wird der *OR*-Modus benutzt [RSS09]. Das bedeutet, der Kompositionsfilter liefert NFE-Prozessbausteine, welche entweder

- mit der ausgewählten Kategorie verknüpft sind oder
- mit irgendeiner Kategorie verknüpft sind, die unterhalb der ausgewählten liegt.

### 7.3 Übersetzung der statischen in ausführbare Vorgehensmodelle

Die Übersetzung der statischen NFE-spezifischen Vorgehensmodelle in ausführbare Modelle erfolgt über die Export-Schnittstelle der PROKRIS-Bibliothek. Diese bildet Modelle im erweiterten SPEM 2.0-Profil auf Modelle in der Sprache aPDL ab. Die Umsetzung des Exports wurde prototypisch in den EPF Composer integriert [RSS09]. Die für die Übersetzung notwendigen Abbildungen zwischen den Modellelementen der beiden Sprachen wurden bereits in Kapitel 6, Tabelle 6.1 definiert.

#### Verknüpfung des Handbuchs

Die Einbindung des Handbuchs in die Ausführungsumgebung wird durch die Integration des entsprechenden Verweiseintrags in die ausführbare Prozessbeschreibung realisiert (siehe Kapitel 5, Listing 5.1). Ermöglicht wird dies durch den gleichzeitigen Export des aPDL-Modells und des Handbuchs.

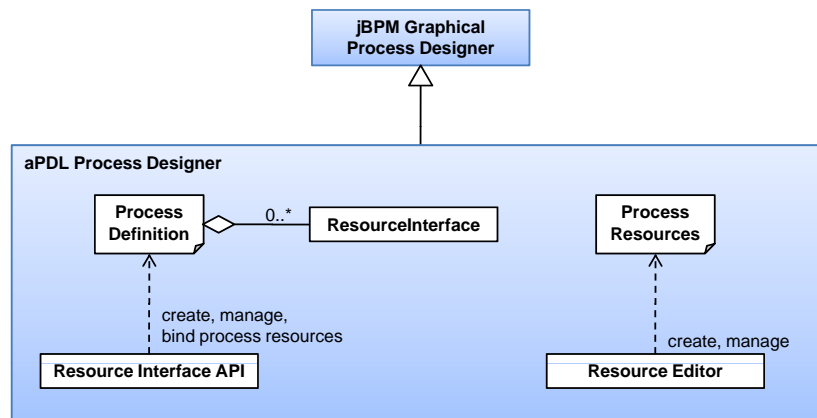


Abbildung 7.4: Architektur des aPDL Designers

Abbildung 7.3 zeigt einen Auszug aus der Architektur der Export-Schnittstelle im Bereich der Menüfunktionen der Oberfläche. Die Klasse `ExportAPDLWizard` stellt mit der Methode `exportAPDL()` die Verbindung zur Logik der aPDL-Codegenerierung her. Die Generierung wird durch Einstellungen in verschiedenen Menüs beeinflusst. `SelectExportDirectoryPage` dient zur Auswahl des Speicherortes und `SelectExportOptionsPage` zur Einstellung verschiedener Parameter. Dazu zählt unter anderem die integrierte Veröffentlichung des Handbuchs, welche über die Klasse `SelectPublishingOptions` gestartet wird. Die Beschreibung im Attribut `description` wird direkt aus der SPEM-Datenstruktur, speziell der Beschreibung des Elements `Step`<sup>1</sup> ausgelesen.

Die beschriebene Funktionalität ist Bestandteil der aPDL-Export-Schnittstelle des EPF Composers [RSS09]. Auch bei mehrfacher Veröffentlichung des Handbuchs (z.B. nach Änderungen) bleiben die Verweisinformationen konsistent. Nur durch das Erstellen neuer SPEM-Elemente entstehen zusätzliche Verweise durch neue Einträge im Handbuch.

## 7.4 Individualisierung

Die ausführbaren Modelle werden im dritten Anpassungsschritt individualisiert. Die notwendige Funktionalität wurde prototypisch innerhalb des aPDL Designers umgesetzt. Im Kapitel 5 waren bereits Abbildungen der Oberfläche des aPDL Designers zu sehen. Im Folgenden wird kurz auf dessen Architektur eingegangen.

### Werkzeugunterstützung

Da die Sprache aPDL auf jPDL aufbaut, ist es naheliegend, den existierenden Editor für jPDL zu erweitern. Abbildung 7.4 gibt einen schematischen Überblick über die Architektur. Der aPDL Designer erweitert den bestehenden Editor um zwei wesentliche Schnittstellen, den *Resource Editor* und die *Resource Interface API*. Erster dient der

<sup>1</sup>Ein `Step` in SPEM wird auf eine `task-node` in aPDL abgebildet (siehe Kapitel 6, Tabelle 6.1).

Spezifikation und Verwaltung der konkreten Ressourcen und wurde bereits in Kapitel 6.3 besprochen. Der Editor ist als Eclipse Plugin in den aPDL Designer eingebunden. Die Schnittstelle *Resource Interface API* stellt die notwendige Funktionalität zur statischen Ressourcenbindung bereit. Neben dem eigentlichen Binden muss der Editor außerdem um das Konzept der Ressourcenschnittstellen erweitert werden. Die Implementierung der Schnittstelle folgt dem *Model-View-Controller*-Prinzip. Sie wurde in [Mut07] umgesetzt und in [RSS09] erweitert. Daher wird für Details auf diese beiden Arbeiten verwiesen<sup>2</sup>.

### 7.5 Zusammenfassung

Das Kapitel stellte sowohl die notwendigen Metamodellkonzepte als auch die technische Umsetzung des PROKRIS-Komposers innerhalb des PROKRIS-Frameworks vor. Im Prozessmetamodell wurde das Konzept des Kompositionsfilters definiert. Bedingt durch die Differenzierung der Sprachkonzepte für statische und ausführbare Vorgehensmodelle auf der Ebene M1 wurden zwei prototypische Werkzeuge zur Bereitstellung der Funktionalität des PROKRIS-Komposers entwickelt. Deren Architekturen wurden in diesem Kapitel kurz vorgestellt. Außerdem wurde auf die Übersetzung der statischen in ausführbare Vorgehensmodelle und die Verknüpfung des Handbuchs mit der ausführbaren Spezifikation, eingegangen.

---

<sup>2</sup>Das Konzept der Ressourcenschnittstelle (Resource Interface) wird in beiden Arbeiten als *Hook* bezeichnet.

## 8 Ausführungsumgebung: Erweiterte Workflowumgebung für PROKRIS-Prozessmodelle

Dieses Kapitel stellt die Ausführungsumgebung des PROKRIS-Frameworks vor. Den Kern der Umgebung bildet ein Workflow Management System. Derartige Systeme sind auf den Einsatz für Geschäftsprozesse spezialisiert. Daher werden zu Beginn die notwendigen Erweiterungen für deren Anwendung zur Unterstützung von Softwareentwicklungsprozessen diskutiert. Im Anschluss daran wird die Architektur der PROKRIS-Ausführungsumgebung vorgestellt und wesentliche Aspekte der Ressourcenzuordnung und damit zusammenhängend die Initiierung von Werkzeugaufrufen erörtert. Abschließend werden interessante Aspekte der Ausführungsunterstützung detaillierter erläutert. Dazu gehören die dynamische Laufzeitzuordnung der Ressourcen, die grafische Benutzerschnittstelle sowie die Unterstützung paralleler Prozesse.

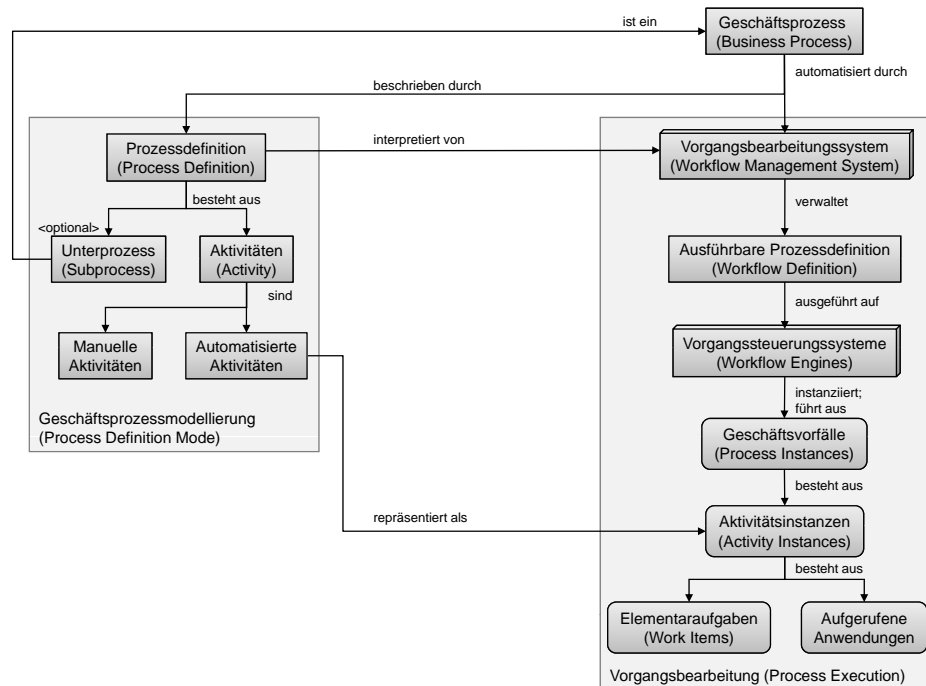
### 8.1 Erweiterung der Standardarchitektur der WfMC

Die Ausführungsumgebung basiert auf der Architektur eines Workflow Management Systems (WfMS). Der ursprüngliche Einsatzbereich von WfMS ist die Prozesserstellung und Ausführung von Geschäftsprozessen. Abbildung 8.1(a) stellt das Vorgehen, wie es die WfM-Coalition definiert, dar. Ausgangspunkt sind die *Geschäftsprozesse* eines Unternehmens. Während der *Geschäftsprozessmodellierung* werden diese unter dem Gesichtspunkt der automatisierten Ausführung innerhalb einer *Prozessdefinition* modelliert. Die Prozessdefinition kann neben *automatisiert* und *manuell* ausführbaren *Aktivitäten* auch aus weiteren Unterprozessen aufgebaut sein. Die Prozessdefinition wird vom WfMS während der *Vorgangsbearbeitung* interpretiert. Die *Workflowmaschine* erzeugt durch Interpretation der Prozessdefinition eine ausführbare *Prozessinstanz* und führt diese aus.

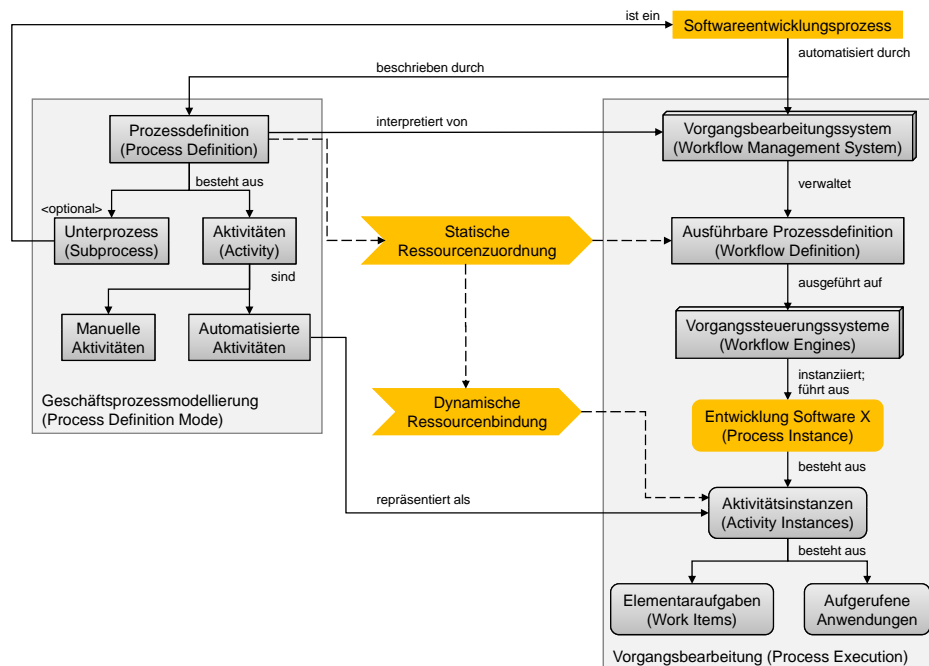
Dieses Vorgehen ist prinzipiell auf die Unterstützung von Softwareentwicklungsprozessen abbildbar, da es sich dabei ebenfalls um die geordnete Abarbeitung von Aktivitäten unter Nutzung von Daten handelt. Statt Geschäftsprozessen (Abbildung 8.1(a)) werden Softwareentwicklungsprozesse (Abbildung 8.1(b)) behandelt. Demnach handelt es sich bei der Prozessinstanz nicht mehr um Geschäftsvorfälle sondern um ein Entwicklungsprojekt.

Zwei Besonderheiten sind nicht direkt umsetzbar. Die erste ist die Möglichkeit der dynamischen Ressourcenbindung zur Laufzeit, wie sie bereits in den vorangehenden Kapiteln erörtert wurde. Diese Funktionalität soll es den Entwicklern ermöglichen, die durch die Ausführungsumgebung bereitgestellte Entwicklungsumgebung an eigene Präferenzen

## 8 Ausführungsumgebung: Erweiterte Workflowumgebung für PROKRIS- Prozessmodelle



(a) Konzepte und deren Beziehungen beim Einsatz eines WfMS für Geschäftsprozesse nach der WfMC (nach [Lie06])



(b) Abbildung der Konzepte auf den Einsatz für Softwareentwicklungsprozesse und notwendige Erweiterungen zur Unterstützung der Ressourcenbindung

Abbildung 8.1: Abbildung der Konzepte der Geschäftsprozessmodellierung und -ausführung auf Softwareentwicklungsprozesse

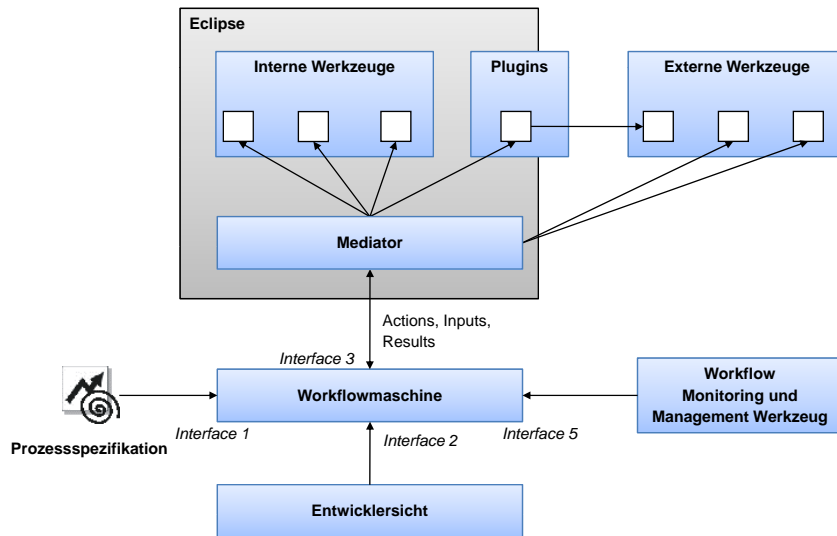


Abbildung 8.2: Architektur des PROKRIS-WfMS mit Schnittstellen der WfMC

anzupassen. Um dieses Konzept zu unterstützen, ist der in der Abbildung dargestellte Ablauf an zwei Stellen zu erweitern (Abbildung (8.1(b))). So werden in die Prozessbeschreibung Gruppen von Ressourcen gebunden (*Statische Ressourcenbindung*). Zur Laufzeit des Prozesses kann durch den Entwickler beim Betreten einer Aktivität aus dieser Gruppe eine konkrete Ressource ausgewählt werden (*Dynamische Ressourcenbindung*). Steht die konkrete Ressource schon zur Modellierungszeit fest oder kommt nur eine einzige in Frage, kann diese natürlich schon statisch gebunden werden. Die zweite Besonderheit ist die notwendige Initiierung von Werkzeugaufrufen und damit verbunden die Möglichkeit des Aufbaus von automatisch ausführbaren Werkzeugketten. Solche Werkzeugketten werden beispielsweise zur, für den Entwickler transparenten, Integration einer Verifikationsschleife benötigt. Beide Merkmale bedingen eine Erweiterung der Standardarchitektur von Workflow Management Systemen der WfM-Coalition.

Abbildung 8.2 gibt einen schematischen Überblick über den erweiterten Aufbau. Kern der Ausführungsumgebung ist eine Workflowmaschine, die Prozessbeschreibungen verwaltet und steuert. Dies entspricht der Schnittstelle (Interface) 1 der Standardarchitektur der WfMC (Kapitel 2.9, Abbildung 2.18). Die Prozessbeschreibung enthält Angaben über Aktivitäten, verantwortliche Rollen und benötigte Werkzeuge. Die Schnittstelle ist so erweitert, dass diese Informationen ausgewertet und daraus Werkzeugketten aufgebaut werden können. Durch Erweiterung der Schnittstelle 3 kann die Workflowumgebung das automatische Aufrufen von Werkzeugen initiieren und den Datenfluss innerhalb der Werkzeuge steuern. Außerdem ist es dem Entwickler durch die erweiterte Funktionalität der Schnittstelle 2 möglich, die dynamische Ressourcenbindung vorzunehmen, sowie an vordefinierten Entscheidungspunkten den Ablauf zu beeinflussen.

Während die Umsetzung der Schnittstellen 1, 2 und 5 durch Erweitern der existierenden Konzepte möglich ist, muss Schnittstelle 3 dagegen neu konzipiert werden. Ziel

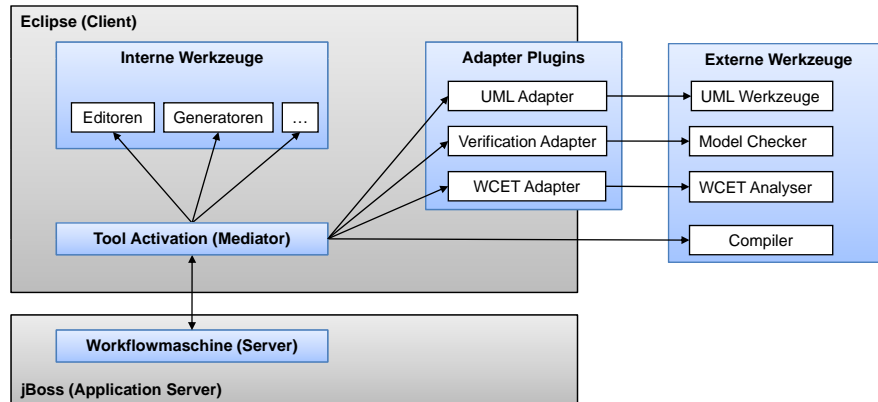


Abbildung 8.3: Beispielarchitektur für die Integration von Werkzeugen der SuReal-Kette

des PROKRIS-Ansatzes ist der Aufbau von Werkzeugketten aus einer flexibel erweiterbaren Menge von Werkzeugen. Allerdings ist die Ansteuerung von Werkzeugen durch Workflowmaschinen bisher nur bedingt möglich. Es existieren einige Systeme, in denen eine festgelegte Menge von Werkzeugen in Arbeitsabläufe integriert werden kann. Die Auswahl der Werkzeuge ist jedoch von vornherein beschränkt ([Har07], S.30).

Die PROKRIS-Ausführungsumgebung implementiert daher das Entwurfsmusters *Mediator*, um die flexible Integration beliebiger Werkzeuge zu ermöglichen (Abbildung 8.2). Das Datenmodell der Werkzeuge und dessen Integration in die Prozessbeschreibung wurde bereits erläutert. Abbildung 8.3 zeigt den Einsatz des Mediators am Beispiel der Werkzeugkette, welche sich aus dem SuReal-Vorgehensmodell ergibt. Dargestellt ist die Palette der Werkzeuge und die Art ihrer Einbindung.

Zusammenfassend sind im Folgenden die zusätzlichen Aufgaben der einzelnen Schnittstellen aufgeführt:

#### Schnittstelle 1:

- Erweiterung der ausführbaren Prozessdefinition um die Möglichkeit des Bindens von konkreten Ressourcen bzw. Ressourcengruppen an abstrakte Ressourcenbeschreibungen
- Bereitstellen einer Prozessdefinition mit zugehöriger Ressourcenspezifikation in der Ausführungsumgebung

#### Schnittstelle 2/5:

- Bereitstellen der Aufgabenlisten zu den einzelnen Projekten. In der Nutzerschnittstelle (2) sind die für den jeweiligen Nutzer relevanten Aufgaben geeignet zu markieren.
- Bereitstellung von Details zu den Aufgaben (z.B. Name, Startdatum, Deadline, Beschreibung, Verknüpfung zum Handbuch)



- Bearbeiten von Entscheidungspunkten zum weiteren Verlauf
- Auswahl von Ressourcen aus Ressourcengruppen: Dazu gehört sowohl die Zuordnung einer Aufgabe zu einer konkreten Person, als auch die Auswahlmöglichkeiten von Werkzeugen und Daten aus einer entsprechenden Gruppe

### Schnittstelle 3:

- Laufzeitbindung der Ressourcen entsprechend der Auswahl der konkreten Ressourcen in Schnittstelle 2/5
- Initiieren von Werkzeugaufrufen mit den zugeordneten Arbeitsdaten
- Unterstützung automatisch ablaufender Werkzeugketten

## 8.2 Die Architektur der PROKRIS-Ausführungsumgebung

Abbildung 8.4 zeigt die Architekturkomponenten der Ausführungsumgebung. Die entsprechende Zuordnung zur Nummerierung der Schnittstellen ist angegeben.

Wie bereits beschrieben, sind durch die automatische Werkzeugintegration und -steuerung sowie die Unterstützung einer flexiblen Ressourcenverwaltung Erweiterungen in der Prozessbeschreibung und der Prozessabarbeitung notwendig. Das Eingabeformat der Ausführungsumgebung ist daher die Sprache aPDL (Schnittstelle 1). Die Umsetzung dieser Schnittstelle ist der aPDL-Designer. Dieser ist Teil des PROKRIS-Komposers und setzt den Schritt der Individualisierung innerhalb der PROKRIS-Methodik um.

Die jBPM-Workflowmaschine stellt als Kern der Ausführungsumgebung zusätzlich einen Java Application Server (JBoss) zur Verfügung. jBPM verwendet über Hibernate eine Datenbank, in der alle Daten zu eingespielten Prozessdefinitionen, instanziierten Prozessen sowie Prozessressourcen stehen. Die zuvor beschriebenen Erweiterungen der Schnittstellen 2/5 und 3 haben nicht nur Auswirkungen auf die Architektur des Clients. Die erweiterte Abarbeitungslogik der ausführbaren Prozesse wird serverseitig durch eine EJB-Komponente, die **Process Manager Bean** bereitgestellt.

In aPDL werden Prozesse als gerichteter Graph modelliert. Um das Knotenverhalten individuell zu beeinflussen können **ActionHandler** definiert werden. In der PROKRIS-Umgebung ist der **ActionHandler** so umgesetzt, dass er die Laufzeitbindung der Ressourcen übernimmt. Für die Schnittstelle 3 gehört dazu das Auslesen der Informationen zu den Werkzeugen aus den Parameterlisten, für die Schnittstelle 2/5 die Informationen zu Personen und deren Rollenzuordnung. Diese Informationen zum aktuellen Prozesszustand speichert der **StateManager**. Die Weiterführung des Prozesses erfolgt je nach Knotentyp durch die automatische oder nutzergesteuerte Weitergabe des Tokens an eine abgehende Transition.

Die Umsetzung der Funktionalität der Schnittstellen 2/5 übernimmt die **Process Manager View**. Über dieses Eclipse Plugin können mehrere Entwickler jeweils über ihre Eclipse-Anwendung (Client) die **Process Manager Bean** ansprechen. Diese Schnittstelle ermöglicht eine Sicht auf die Prozesse in der Datenbank sowie Benutzerinteraktionen,

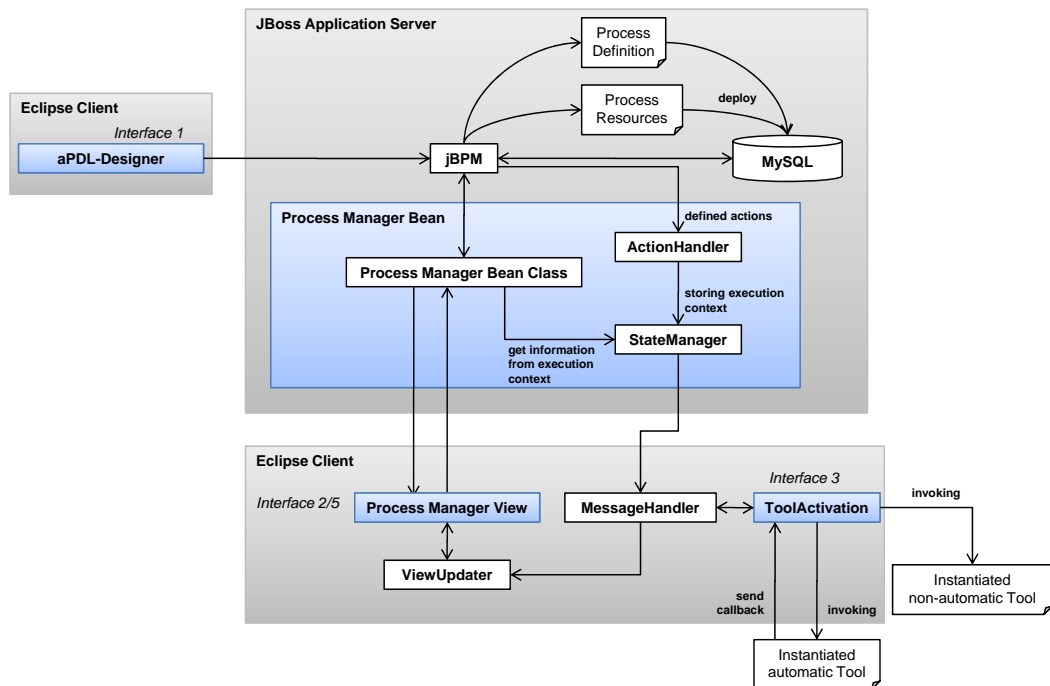


Abbildung 8.4: Architektur der PROKRIS-Ausführungsumgebung

wie z.B. das Abschließen von Aufgaben und Fortschreiten im Prozess. Der Unterschied in den beiden Schnittstellen besteht in der Bereitstellung einer nutzerspezifischen Sicht (Schnittstelle 2) und einer Gesamtansicht aller Prozesse und Prozessinstanzen zum Projektmanagement (Schnittstelle 5).

Die Kommunikation zwischen Client und Server kann sowohl synchron, für Aufrufe und Informationsanfragen zwischen **Process Manager View** und **Process Manager Bean**, als auch asynchron erfolgen. Asynchrone Kommunikation ist für Informationen an den Client durch die **ActionHandler** oder Benachrichtigungen nach Prozessänderungen durch andere Nutzer notwendig.

Schnittstelle 3 ist in der PROKRIS-Umgebung so erweitert, dass die automatische Integration von Werkzeugen möglich ist. Dies erfolgt über das Eclipse Plugin **ToolActivation**, welches als Mediator zwischen Workflowmaschine und den einzelnen Werkzeugen fungiert. Wird an der aktuellen Stelle im Prozess die Beschreibung eines auszuführenden Werkzeuges zurück geliefert, stellt das **ToolActivation**-Plugin aus den Informationen den Werkzeugaufruf zusammen und startet diese Anwendung auf dem Rechner des Entwicklers. Es können sowohl Einzelwerkzeuge (non-automatic Tool) als auch automatisch ausgeführte Werkzeuge (automatic Tool) aufgerufen werden. Automatisch bedeutet, dass ohne Benutzerinteraktion zum nächsten Knoten im Prozess weitergeschaltet wird, wodurch der Aufbau von Werkzeugketten möglich ist. Dazu muss ein **CallBack** als Benachrichtigung an die **ToolActivation** gesendet werden.

Die Architektur entspricht dem Prinzip von *Model-View-Controller*. Die *View* wird

durch die **Process Manager View** realisiert. Der **State Manager** übernimmt die Rolle des *Models*, indem er sowohl die Prozess- und Ressourcendaten aus der Datenbank, als auch die angemeldeten Nutzer verwaltet. Die **ActionHandler** beeinflussen als *Controller* den Ablauf der Prozesse.

## 8.3 Details der prozessbasierten Ausführungsunterstützung

Die Umsetzung der Architektur wirft eine ganze Reihe interessanter Fragen auf, von denen zwei ausgewählte näher betrachtet werden sollen. Diese sind die Oberfläche der Nutzerschnittstelle sowie die Laufzeitzuordnung der Ressourcenbeschreibungen. Ersteres gibt näheren Einblick in die Erweiterung der Schnittstellen 2/5. Das zweite ermöglicht die Diskussion der Erweiterungskonzepte der Schnittstelle 3.

Es wird darauf verzichtet implementierungstechnische Details zu diskutieren. Im Rahmen des SuReal-Projekts erfolgte die prototypische Implementierung der Ausführungsumgebung. Diese ist in [RSS09] dokumentiert. Detaillierte Informationen können daher dort nachgelesen werden.

### 8.3.1 Grafische Benutzerschnittstelle

Die Benutzerschnittstelle der Ausführungsumgebung dient prinzipiell dazu, den Benutzern aktuelle Prozessinformationen anzuzeigen sowie eine Beeinflussung des Prozessablaufes zu ermöglichen. Sie bildet damit den zentralen Entwicklerzugang der prozessbasierten Entwicklungsumgebung. Für deren Umsetzung stehen verschiedene Möglichkeiten zur Verfügung. In der PROKRIS-Ausführungsumgebung wird die Schnittstelle durch ein Eclipse-Plugin, der **Process Manager View**, realisiert. Dies hat den Vorteil, dass dem Benutzer auch visuell eine integrierte Umgebung zur Verfügung steht.

Im Folgenden wird auf das Zusammenspiel von Oberfläche und Prozessverwaltung eingegangen. Weiterhin wird die Darstellung der für den Entwickler relevanten Informationen diskutiert. Abschließend werden die Möglichkeiten der Beeinflussung des Prozessablaufes durch den Entwickler aufgezeigt. Dabei werden zur Veranschaulichung Auszüge aus der prototypischen Implementierung gezeigt. Diese sind unter dem Gesichtspunkt entwickelt worden, einen Machbarkeitsbeweis des PROKRIS-Ansatzes zu liefern. Die Oberfläche ist daher nicht auf Bedienung und Nutzbarkeit optimiert und damit verbesserbar. Außerdem ist die Umsetzung nicht vollständig im Sinne der Anzeige aller notwendigen Informationen und der Bereitstellung der gewünschten Funktionalität.

#### Modell der grafischen Benutzeroberfläche

Die Benutzeroberfläche setzt die *View* der *Model-View-Controller*-Architektur um. Ziel ist es, die relevanten Prozessinformationen in geeigneter Weise darzustellen und Interaktionsmöglichkeiten der Nutzer mit der Ausführungsumgebung anzubieten. Das Modell der Benutzeroberfläche folgt daher dem *Overview-Plus-Detail-Muster* [Tid05]. Abbildung 8.5 zeigt beispielhaft eine Ansicht der Oberfläche. Die Oberfläche ist in zwei Hauptbereiche gegliedert. Im linken Teil wird der Prozessbaum zur Navigation angezeigt (*Overview*), im rechten Teil finden sich detaillierte Informationen zu dem im Baum

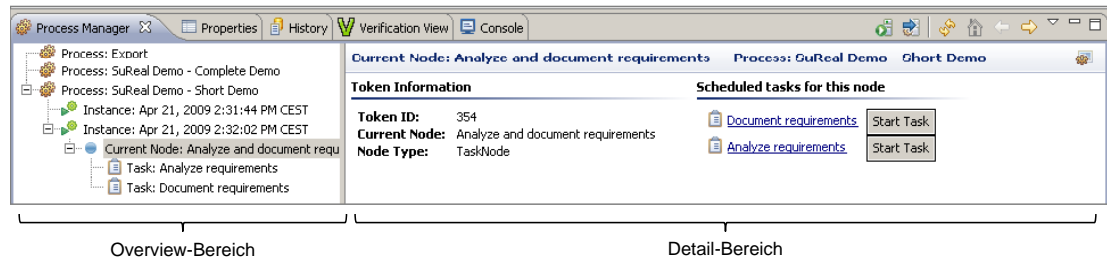


Abbildung 8.5: Prozess Manager View bei ausgewähltem Knoten einer Prozessinstanz sowie Zuordnung der Oberflächenbereiche zum Overview-Plus-Detail-Muster

ausgewählten Eintrag (*Detail*). Der Informationsfluss erfolgt von der Datenbank über das **Process Manager Bean** in die **Process Manager View** und dort vom Prozessbaum zur Anzeige der Details. Der Overview- und Detail-Bereich wird durch das im SuReal-Projekt entwickelte *SheetViewer SWT-Widget* erstellt und verarbeitet [RSS09].

In der Datenbank hinterlegte Prozessinformationen lassen sich in der *Overview*-Ansicht auf verschiedenen Ebenen verwalten. Wie in Abbildung 8.5 zu sehen, sind auf der obersten Ebene alle bereitgestellten Prozessdefinitionen hinterlegt. Von jeder Prozessdefinition können mehrere Instanzen existieren, welche sich in einem unterschiedlichen Zustand befinden können. Dieser wird durch den aktuellen Prozessknoten (*Current Node*) repräsentiert. Auf der untersten Ebene der Prozessverwaltung werden die zu dem jeweiligen Knoten gehörenden Aufgaben angezeigt.

### Bereitstellung detaillierter Prozessinformationen

In der Detailanzeige der Oberfläche werden äquivalent zum *Overview* vier Stufen des Detaillierungsgrades unterschieden: Prozessdefinition, Prozessinstanz, Knoten mit zugehörigen Aufgaben sowie Aufgabendetails.

Auf der ersten Stufe der Prozessdefinition sind durch Auswahl einer Prozessdefinition im *Overview* folgende Informationen bzw. Funktionalitäten anzubieten:

- Starten des JBoss Application Servers
- Ablegen einer Prozessdefinition in der Datenbank
- Anzeigen einer Prozessübersicht, so dass der Nutzer einordnen kann, wo er sich im Prozessablauf befindet
- Erstellen einer neuen Instanz des selektierten Prozesses
- Löschen der selektierten Prozessdefinition aus der Datenbank
- Aktualisieren des Prozessbaums
- Hilfsfunktionen zur Navigation im Prozessbaum (z.B. Expandieren)

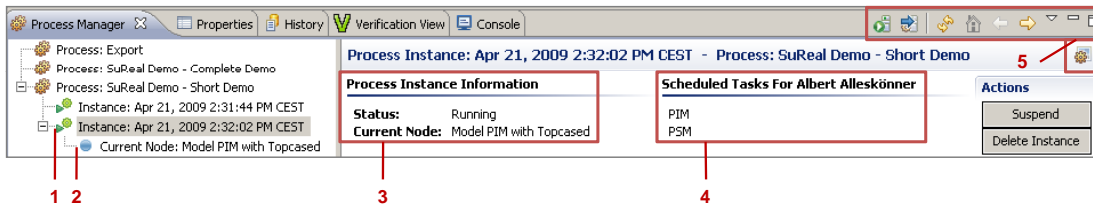


Abbildung 8.6: Prozess Manager View bei selektierter Prozessinstanz

Diese Ansicht sollte nur in der Schnittstelle 5 (Workflow Monitoring und Management) sichtbar sein. Der in ein Projekt involvierte Entwickler erhält nach erfolgreicher Anmeldung am System sofort die Detailstufe der ausgewählten Prozessinstanzen (Abbildung 8.6). Folgende Informationen bzw. Funktionen werden dem Nutzer angeboten:

1. Zeitpunkt der Erstellung der Instanz sowie Statusanzeige durch Symbol (erstellt, gestartet, gestoppt, beendet).
2. Anzeige des aktuellen Knotens
3. Allgemeine Informationen zur Instanz
4. Zugeordnete Aufgaben für die aktuell angemeldete Person
5. Funktionalität zur Beeinflussung der View (z.B. Übersichtsbild, expandieren des Prozessbaums)

In Abbildung 8.6 werden die Aufgaben des aktuell angemeldeten Nutzers in der *Detail*-Sicht angezeigt. Die *Detail*-Sicht kann durch Beschränkung des Prozessbaumes des *Overview*s auf für den Nutzer wichtige Prozesse, Prozessinstanzen und Aufgaben übersichtlicher gestaltet werden.

Während der Bearbeitung einer Prozessinstanz werden die aktuell zu bearbeitenden Aufgaben angezeigt. Abbildung 8.5 zeigte beispielhaft innerhalb einer ausgewählten Prozessinstanz einen Knoten mit zugeordneten Aufgaben. Diese können gestartet werden. Dabei werden automatische und manuelle Aufgaben unterschieden. Letztere müssen explizit durch den Benutzer nach der Bearbeitung beendet werden. Dies ist zum Beispiel bei Modellierungsaufgaben sinnvoll, die zum Teil mehrere Tage in Anspruch nehmen können.

Von dieser Sicht kann man in eine weitere Ansicht, die Aufgaben-Ansicht, wechseln (Abbildung 8.7). Diese enthält detaillierte Informationen über die gewählte Aufgabe, wie Name, Status, ID der zugehörigen Instanz, gebundene Ressourcengruppen, kurze Beschreibung und Link zur Dokumentation. Die beiden letzteren sind der Verknüpfungspunkt zur integrierten Hilfe über das Onlinehandbuch.

Die Auswahl konkreter Ressourcen bei gebundenen Ressourcengruppen (dynamische Ressourcenbindung) erfolgt in der prototypischen Implementierung über Dialoge beim Starten der Werkzeuge. In [Sch07] wurde bereits eine optimalere Auswahl über Check-Boxen

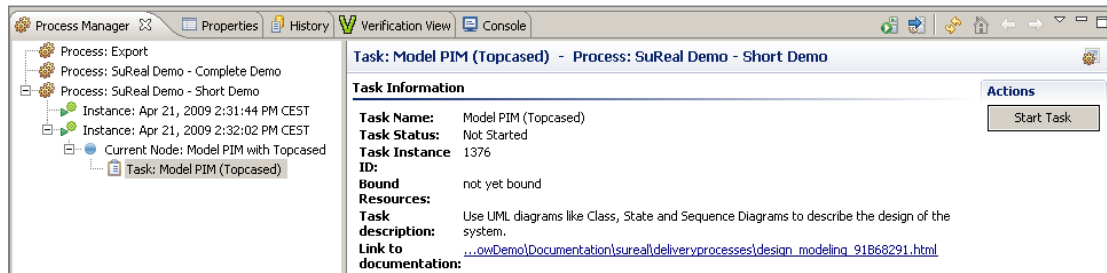


Abbildung 8.7: Detailinformationen zu einer ausgewählten Aufgabe

innerhalb einer Detail-Ansicht entworfen. Außerdem wurde dort der flexible Einsatz der Personengruppen zur Laufzeit eines Projekts (in Form einer Prozessinstanz) untersucht. Die Implementierung erfolgte unter Nutzung von JSP (Java Server Pages) und wurde noch nicht mit der hier vorgestellten Implementierung integriert. Dies sollte aber in weiterführenden Arbeiten umgesetzt werden, um die Gebrauchstauglichkeit zu erhöhen.

Abschließend wird darauf hingewiesen, dass die vorgestellte Umsetzung der Benutzeroberfläche nicht den Anspruch auf Vollständigkeit erhebt. Sie ist flexibel erweiterbar. So sind zum Beispiel zur Unterstützung paralleler Subprozesse innerhalb einer Projektinstanz keine aktuellen Einzelknoten mehr anzuzeigen, sondern eine Liste der Knoten (Token), welche momentan aktiv sind.

### Beeinflussung des Prozessablaufes

Die Beeinflussung des Prozessablaufes durch den Entwickler beschränkt sich nicht nur auf das Starten und Beenden von Aufgaben und die Laufzeitzuordnung der Ressourcen. Bereits bei der Komposition des ausführbaren Prozesses durch den Prozessingenieur bzw. dessen Individualisierung durch den Projektmanager können Entscheidungspunkte durch den Einsatz verschiedener Knotentypen eingebaut werden, an denen der Entwickler während der Projektbearbeitung den Ablauf beeinflussen kann. Bisher wurden als Knoten eines Prozesses vor allem *Task* und *Task-Node* eingesetzt. Es existieren weitere Knotenarten, die in der PROKRIS-Ausführungsumgebung für Entscheidungspunkte verwendet werden können. Diese sind:

*State:* In *State*-Knoten wird die Abarbeitung prinzipiell angehalten. So können sie verwendet werden, falls ein externes System zu benachrichtigen und auf dessen Antwort zu warten ist. Außerdem wird der *State*-Knoten für manuelle Entscheidungen im Prozess eingesetzt. In *State*-Knoten können mehrere ausgehende Transitionen modelliert werden. So kann der Nutzer die gewünschte bzw. notwendige ausgehende Transition und damit den weiteren Prozessverlauf wählen.

*Decision:* Sollen innerhalb eines Prozesses automatische Entscheidungen getroffen werden, so bietet sich der Einsatz von *Decision*-Knoten mit einer zugehörigen *DecisionHandler*-Implementierung an. Diese ermittelt anhand der Prozessdefinition und den Informationen aus dem *CallBack* des Werkzeugs automatisch die einzuschlagende Transition. Alternativ können zur Entscheidungsfindung den ausgehen-

den Transitionen **Condition**-Elemente zugewiesen werden, welche vom *Decision*-Knoten zur Laufzeit ausgewertet werden. In diesem Fall wird der ersten Transition gefolgt, bei welcher die Auswertung der Bedingung **true** liefert.

*Mail Node:* Das automatische Senden von E-Mails aus dem Prozess heraus kann ebenfalls modelliert und unterstützt werden. Damit kann man Aufgaben an andere Projektbeteiligte zuweisen oder Erinnerungen verschicken lassen.

#### 8.3.2 Ressourcenmanagement zur Laufzeit

Das Ressourcenmanagement zur Laufzeit, das heißt, die Verwaltung und dynamische Bindung von aufgabenspezifischen konkreten Ressourcen, ist ein wesentliches Ziel des PROKRIS-Ausführungssystems. Es hat zwei Aufgaben zu erfüllen. Zum Einen müssen die Ressourcendaten über die Benutzeroberfläche zugänglich gemacht werden. Zum Anderen wurden die konkreten Ressourcen während der Individualisierung nur durch Referenzen auf Einträge in der separaten Ressourcenspezifikation gebunden. Diese Referenzen sind während der Ausführung aufzulösen. Die Ressourcenspezifikation kann so parallel zur Prozessspezifikation in der Datenbank oder verteilt verwaltet werden.

Es kann an dieser Stelle nicht auf alle Aspekte der Erweiterung der entsprechenden Schnittstelle 3 der WfMS-Architektur eingegangen werden. Für weitere Details wird auf [RSS09] verwiesen.

##### Laufzeitbindung

Die Umsetzung der dynamischen Ressourcenbindung zur Prozesslaufzeit erfolgt über die Komponente **Assignment**. Deren Umsetzung basiert teilweise auf [Sch07] und [Har07]. Abbildung 8.8 zeigt einen Auszug aus der Klassenstruktur. Die wesentliche Klasse ist der **RessourceAssignmentHandler**, eine Implementierung der Schnittstelle **ActionHandler** (Abbildung 8.4) der jBPM-Workflowmaschine. Zudem beinhaltet diese Komponente weitere Klassen, welche jeweils die Bindung einer bestimmten Ressource an eine Ressourcenschnittstelle übernehmen. Im Folgenden werden die einzelnen Klassen vorgestellt:

**RessourceAssignmentHandler:** implementiert einen **ActionHandler** (Interface `org.jbpm.graph.def.ActionHandler`) als Schnittstelle zur jBPM-Maschine. Die Methode `execute()` wird durch eine Referenz in der Prozessbeschreibung aufgerufen und startet die Verarbeitung der mitgelieferten Informationen über die Ressourcenschnittstellen. Zu jeder Schnittstelle wird eine Unterklasse von **ResourceBinder** erzeugt und ausgeführt. Abbildung 8.8 zeigt einen Auszug aus einer Prozessdefinition mit dem Tag `<action/>` als Referenz.

**ResourceBinder:** stellt als abstrakte Oberklasse einer Klassenhierarchie abstrakte Funktionen zum Binden konkreter Ressourcen bereit.

**RoleBinder, ToolBinder, MaterialBinder:** erben von **ResourceBinder** und realisieren das Auflösen der Referenzen und tatsächliche Binden der konkreten Ressourcen. Für jeden Ressourcentyp existiert eine Klasse.

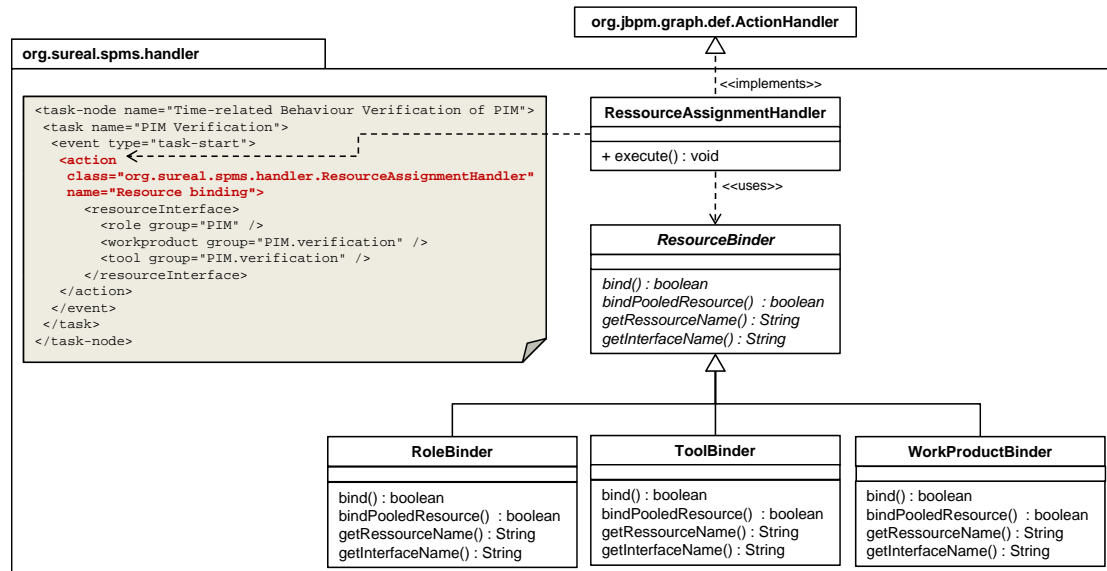


Abbildung 8.8: Vereinfachte Klassenstruktur der Implementierung des ActionHandlers zur Ressourcenbindung

Wurde eine Ressourcengruppe definiert, werden dem Nutzer alle Ressourcen aus der Gruppe zur Auswahl angeboten. Erst danach wird z.B. bei einem Werkzeug der konkrete Aufruf mit den entsprechenden Parametern und Datenquellen zusammengestellt und ausgeführt.

### Erweiterbarkeit

Das Ressourcenmanagement wurde durch die Nutzung hierarchischer Strukturen so entworfen, dass es sowohl um neue Daten bestehender Ressourcen als auch um neue Typen erweiterbar ist. Ein weiterer Vorteil für Verbesserung, Erweiterung und Wartung ist der Einsatz der Model-View-Controller-Architektur.

## 8.4 Zusammenfassung

Dieses Kapitel stellte die PROKRIS-Ausführungsumgebung vor, welches auf den Konzepten eines WfMS beruht. Diese werden vorwiegend zur Unterstützung von Geschäftsprozessanwendung eingesetzt. Daher wurden zu Beginn die notwendigen Erweiterungen der Standardarchitektur eines WfMS, wie es die WfM-Coalition definiert, motiviert und spezifiziert. Erweiterungen sind in folgenden Schnittstellen notwendig: Schnittstelle 1: *Process Definition Tools*, Schnittstelle 2/5: *Workflow Client/ Management Application* und Schnittstelle 3: *Invoked Applications*. Darauf basierend wurde die Architektur des PROKRIS-Systems erläutert. Abschließend wurden beispielhaft zwei Aspekte der Umsetzung näher diskutiert, die grafische Benutzerschnittstelle und das Ressourcenmanagement zur Laufzeit.



Diese Arbeit leistet damit einen wesentlichen Beitrag zur Verknüpfung der Modellierung mit der Ausführung. Innerhalb der Ausführung entstehen neue Forschungsfragen, auf die im Ausblick (Kapitel 11) eingegangen wird.



## 9 Evaluierung von PROKRIS-Methodik und -Framework

In den vorangegangenen Kapiteln wurden verschiedene NFE-Prozessmuster und deren Integration in Vorgehensmodelle auf Basis der PROKRIS-Methodik definiert. Diese Arbeit stellt damit erstmals einen systematischen Ansatz zur konsequenten Unterstützung laufzeitkritischer nicht-funktionaler Anforderungen durch Vorgehensmodelle vor. Von existierenden Vorgehensmodellen unterscheidet sich dieser Ansatz sowohl durch die Bereitstellung von NFE-Prozessmustern als auch der starken Ausrichtung auf die Wiederverwendung von Prozessbausteinen und der Verknüpfung der Abstraktionsebenen von Vorgehensmodellen.

Die Anwendbarkeit der entwickelten methodischen und technischen Konzepte des PROKRIS-Ansatzes ist durch den praktischen Einsatz in konkreten Projekten zu überprüfen. Ein solches Projekt ist die Entwicklung des Vorgehensmodells innerhalb des SuReal-Projekts, welches bereits innerhalb der Arbeit als durchgängige Fallstudie verwendet wurde. Das Vorgehensmodell wurde unter Einsatz der entwickelten prototypischen Werkzeugumgebung modelliert, an Realzeitanforderungen angepasst sowie die daraus resultierende prozessbasierte Werkzeugumgebung in einem konkreten Entwicklungsprojekt eingesetzt.

Die Bewertung des PROKRIS-Ansatzes folgt den in Kapitel 1 aufgestellten Kriterien. Die Fallstudie des SuReal-Projekts bildet den Kern dieser Evaluierung. Die PROKRIS-Methodik beruht auf dem Einsatz verschiedener NFE-Prozessmuster zur Umsetzung nicht-funktionaler Anforderungen innerhalb von Vorgehensmodellen. Bevor die Methodik an sich evaluiert wird, wird daher anhand der in Kapitel 4.2 analysierten notwendigen Eigenschaften eines derartigen Vorgehensmodells überprüft, ob diese durch die NFE-Prozessmuster und die PROKRIS-Methodik erfüllt werden.

### 9.1 Gewährleistung der Eigenschaften von Vorgehensmodellen zur Entwicklung laufzeitkritischer Software

In Kapitel 4.2 wurden Eigenschaften eines Vorgehensmodells analysiert, welches die Entwicklung laufzeitkritischer Softwaresysteme unterstützt. Anhand der folgenden Gegenüberstellung dieser Eigenschaften mit den verschiedenen Konzepten des PROKRIS-Ansatzes wird gezeigt, dass ein durch die NFE-Prozessmuster erweitertes Vorgehensmodell und den Einsatz der damit verbundenen prozessbasierten Ausführungsumgebung die geforderten Eigenschaften erfüllt werden.

**Allgemeine Eigenschaften (0.1-0.5):** Die Erfüllung dieser Anforderungen ist durch fol-

gende Konzepte der PROKRIS-Methodik gewährleistet.

*Dokumentierbarkeit (0.1), Auditierbarkeit (0.3):* Die Dokumentierbarkeit und Auditierbarkeit des Vorgehensmodells wird durch die gesteuerte Ausführung der Aktivitäten und Werkzeuge in der prozessbasierten Ausführungsumgebung ermöglicht.

*Standardisierung (0.2):* Die Standardisierung wird durch den Einsatz der NFE-Prozessmuster sowie der Bereitstellung einer einheitlichen Beschreibungssprache erreicht.

*Verifizierungs- und Validierungsaktivitäten (0.4):* Durch Integration der NFE-Prozessmuster werden die notwendigen Aktivitäten, Arbeitsprodukte und Rollen zur Umsetzung der Verifizierungs- und Validierungsaktivitäten in einem Vorgehensmodell etabliert.

*Robustheit (0.5):* Die Robustheit in Bezug auf das Scheitern einzelner Prozessaktivitäten ist im betrachteten Anwendungsfeld vor allem im Bereich der Verifikation und Validierung herzustellen. Erreicht wird dies zum Einen durch das neue Konzept der *Additiven Analyse* (Kapitel 4.3, Abbildung 4.7), welches sich ergänzende Überprüfungen von NFE fordert. Zum Anderen ist durch den Einsatz der iterativen Entwicklung sowie der damit verbundenen Überprüfung der globalen nicht-funktionalen Anforderungen ein mehrmaliger Durchlauf von Verifikations- und Validierungsaktivitäten innerhalb einer Entwicklung vorgesehen. Alle drei Konzepte sind in entsprechenden NFE-Prozessmustern beschrieben (A.5, A.9 und A.11)).

**Phasenübergreifende Eigenschaften (1.1-1.7):** Diese Eigenschaften werden durch die Integration der folgenden NFE-Prozessmuster in das erweiterte Vorgehensmodell erzielt.

*Explizite Spezifikation der NFE (1.1):* NFE-Prozessmuster „Explizite Spezifikation der NFE“ (A.1) und „NFE-spezifische Sprache“ (A.12)

*Rückverfolgbarkeit der NFE (1.2):* NFE-Prozessmuster „NFE-spezifische Abbildungsbeschreibung“ (A.13) und „Transformation der NFE-Spezifikation“ (A.6)

*Analyse der gegenseitigen Beeinflussung der NFE (1.3):* NFE-Prozessmuster „Konfliktanalyse“ (A.3) und „Analyse der Operationalisierung“ (A.7)

*Kontinuierliche Überprüfung der NFE (1.4):* NFE-Prozessmuster „Additive Verifikation der NFE“ (A.5)

*Überwachung der globalen nicht-funktionalen Anforderungen (1.5):* NFE-Prozessmuster „Komposition der NFE“ (A.8)

*Entwicklung entlang des kritischen Szenarios (1.6):* NFE-Prozessmuster „Entwicklung entlang des kritischen Pfades“ (A.10)

*Inkrementelle Entwicklung kritischer Systembereiche (1.7):* NFE-Prozessmuster „Inkrementelle Entwicklung der kritischen Systembereiche“ (A.11)

**Phasenspezifische Eigenschaften:** Durch den Einsatz der NFE-Prozessmuster ist die Erfüllung der geforderten Eigenschaften eines Vorgehensmodells zur Entwicklung laufzeitkritischer Softwaresystem gewährleistet. Dies wurde bereits in Kapitel 4.3 diskutiert. Die verschiedenen NFE-Prozessmuster sind entweder phasenunabhängig definiert oder in ihrer Beschreibung wird auf die unterschiedliche Ausprägung der Muster für den Einsatz in den einzelnen Entwicklungsphasen eingegangen.

### Bewertung

Der PROKRIS Ansatz und insbesondere die definierten NFE-Prozessmuster stellen die Erfüllung der Eigenschaften, welche für Vorgehensmodelle zur Entwicklung laufzeitkritischer Eigenschaften aufgestellt wurden, sicher. Ein Beispiel für die Integration der NFE-Prozessmuster in ein abstraktes NFE-Vorgehensmodell enthält Anhang D. Es handelt sich dabei um das abstrakte NFE-Vorgehensmodell der SuReal-Fallstudie.

Die NFE-Prozessmuster spiegeln den aktuellen Stand an Expertenwissen wieder und sind daher in diesem Sinne niemals vollständig. Durch die Anwendung in konkreten Projekten und deren Vorgehensmodellen kann daher sowohl eine Verbesserung als auch die Erweiterung der NFE-Prozessmuster erreicht werden.

Die NFE-Prozessmuster sind vollständig mittels der in dieser Arbeit vorgestellten Erweiterung von SPEM 2.0 modellierbar und liegen als prototypische Bibliothek innerhalb des PROKRIS-Frameworks vor.

## 9.2 Fallstudien zu NFE-spezifischen Vorgehensmodellen

Durch den Einsatz der PROKRIS-Methodik in Fallstudien kann sowohl die Durchgängigkeit der Vorgehensmodellierung als auch die Anwendung auf verschiedene Arten nicht-funktionaler Eigenschaften gezeigt werden. Die Schwierigkeit der fehlenden Literatur zu Vorgehensmodellbeschreibungen, die für die Entwicklung laufzeitkritischer Anwendung erfolgreich eingesetzt werden konnten, wurde bereits mehrfach in der Arbeit angesprochen. Daher wurde für die Bewertung der Anwendbarkeit der PROKRIS-Methodik einerseits die Fallstudie des SuReal-Projekts, und damit ein *realzeitspezifisches NFE-Vorgehensmodell*, erstellt. Eine zweite Anwendung der Methodik wurde durch die Anpassung des abstrakten NFE-Vorgehensmodell an ein *sicherheitspezifisches NFE-Vorgehensmodell*, wie es in [Pop05] beschrieben ist, realisiert. Beide spezifischen NFE-Vorgehensmodelle sind in Anhang D dargestellt. Um die Durchgängigkeit bis zum ausführbaren Prozessmodell zu demonstrieren, wurde das realzeitspezifische Vorgehensmodell anhand des in Anhang C spezifizierten Ressourcenmodells individualisiert. Das ausführbare Modell ist ebenfalls in Anhang D dargestellt.

### Bewertung

Die Fallstudie zum realzeitspezifischen NFE-Vorgehensmodell demonstriert die Durchgängigkeit der Methodik von den NFE-Prozessmustern und dem abstrakten NFE-Vorgehensmodell bis zur ausführbaren Prozessspezifikation. Während im Anhang D die einzelnen Modelle dargestellt sind, wurde im Kapitel 5 die Anwendung der Methodik und ih-

rer Anpassungsschritte erläutert. Die Fallstudien zeigen außerdem, dass die Mächtigkeit der definierten Sprachmittel für die Anwendung der Methodik ausreichend ist.

Die Anwendung des Konkretisierungsschrittes nicht nur auf das für die SuReal-Fallstudie eingesetzte realzeitspezifische Vorgehensmodell sondern auch auf ein sicherheitspezifisches NFE-Vorgehensmodell zeigt weiterhin, dass die PROKRIS-Methodik für Projekte mit unterschiedlichen nicht-funktionalen Laufzeitanforderungen möglich ist. Voraussetzung dafür ist die Bereitstellung entsprechender NFE-spezifischer Bausteine in der PROKRIS-Prozessbibliothek.

Die Anwendung der PROKRIS-Methodik auf die Fallstudien zeigte einige weiterführende Probleme auf. So wird die Abhängigkeit von NFE-Prozessmustern untereinander nur unzureichend unterstützt (z.B. die Abhängigkeit der *Additiven Verifikation* mit dem dafür notwendigen mehrfachen Einsatz des Musters *Verifikationsschleife*). Dieses Wissen ist nur implizit in der Musterbeschreibung vorhanden und kann so nicht durch den Kompositionsfilter explizit ausgewertet werden. Außerdem wurde die Struktur der konkreten Ressourcen vereinfacht modelliert. Zu einer überzeugenden Anwendung in der Praxis ist die Methodik in diesem Bereich auf der Basis existierender Organisations-Metamodelle zu erweitern.

### 9.3 Kombination mit anderen Softwareentwicklungsprozessen

Für die Anwendbarkeit der PROKRIS-Methodik in der Praxis ist die Möglichkeit der Integration in existierende Softwareentwicklungsprozesse wesentlich. Daher wird im Folgenden beispielhaft der Einsatz des PROKRIS-Frameworks in Kombination mit dem V-Modell XT, dem Rational Unified Process (RUP), der agilen Entwicklung und der modellgetriebenen Entwicklung betrachtet.

#### V-Modell XT

In Kapitel 4.1 wurde analysiert, dass das V-Modell XT Aktivitäten und Produkte für qualitätssichernde Maßnahmen nur allgemein beschreibt und eine systematische Unterstützung für die Anpassung an laufzeitkritische Anforderungen nicht Inhalt des V-Modells XT ist. Daher ist es interessant, die Integration der PROKRIS-Methodik in das V-Modell XT zu betrachten.

Die Integration der ersten beiden Schritte, Tailoring und Konkretisierung, ist konzeptionell problemlos möglich, da das V-Modell XT sowohl wiederverwendbare Prozessbausteine (Vorgehens- bzw. Ablaufbausteine) als auch das Einfügen optionaler Vorgehensbausteine als Form eines Tailoring-Konzepts unterstützt [HH08]. Um auch eine Integration auf technischer Ebene zu erreichen, ist es notwendig das PROKRIS-Metamodell auf das Metamodell des V-Modells XT abzubilden oder umgekehrt. Prinzipiell sind die Begrifflichkeiten und Modellierungskonzepte beider Metamodelle ähnlich, so dass dies ohne größere Probleme möglich sein sollte.

Der dritte Schritt der PROKRIS-Methodik, die Individualisierung, ist nicht direkt auf Konzepte des V-Modells XT abbildbar. Das V-Modell XT kennt zwar eine organisationsspezifischen Anpassung. Diese findet aber noch vor der projektspezifischen Anpassung (welche im Fokus der PROKRIS-Methodik liegt) statt und beschreibt die

Anpassung des V-Modells XT an firmenspezifische Regelwerke. Die Möglichkeit der Integration des Individualisierungsschrittes der PROKRIS-Methodik ist abhängig von der Laufzeitumsetzung des V-Modells innerhalb eines Projekts. Ist eine Unterstützung der Entwicklung durch eine prozessbasierte Entwicklungsumgebung geplant, ist es möglich, die PROKRIS-Ausführungsumgebung dafür einzusetzen. Das V-Modell XT bietet auf der Grundlage einer XML-Struktur des Metamodells alle notwendigen Informationen an [Kuh08], so dass bei entsprechender technischer Anpassung an die Sprache aPDL eine Integration sowohl der Individualisierung als auch der prozessbasierten Werkzeugunterstützung möglich ist.

#### **RUP**

Auch für den RUP wurde in Kapitel 4.1 das Fehlen eines systematischen Vorgehens zur Sicherung nicht-funktionaler Laufzeitanforderungen analysiert. Die PROKRIS-Methodik ist sowohl konzeptionell als auch technisch problemlos in den RUP integrierbar. Dies wurde anhand einer Integration in den Open Unified Process (OpenUP, [Gus08]) getestet. OpenUP ist ein Entwicklungsprozess, der an den RUP angelehnt ist und die grundlegenden Eigenschaften des RUP (siehe Kapitel 2.5) enthält. Für den in dieser Arbeit erweiterten EPF Composer existiert eine Methodenbibliothek für den OpenUP. Diese konnte ohne Weiteres mit der Prozessbibliothek und den Techniken der PROKRIS-Methodik verwendet werden. Da auf Basis des EPF Composers ein kommerzielles Werkzeug, der Rational Method Composer [IBM], zur Implementierung von Vorgehensmodellen auf der Basis des RUP existiert, kann davon ausgegangen werden, dass die PROKRIS-Methodik auch in den RUP selbst integriert werden kann.

#### **Modellgetriebene Entwicklung**

Das Paradigma der modellgetriebenen Entwicklung (Model Driven Development, MDD) ist unabhängig von einer Integration in einen spezifischen Entwicklungsprozess definiert. Eine grundlegende Voraussetzung für den Einsatz der MDD in einem Entwicklungsprojekt ist ein Prozess- bzw. Vorgehensmodell, das seinen Fokus auf Modelle und Transformationen legt. Prinzipiell kann die MDD daher sowohl in agilen Projekten als auch in Projekten mit „schweren“ Prozessen eingesetzt werden [KWB03].

Die innerhalb des PROKRIS-Ansatzes definierten NFE-Prozessmuster beschreiben im Kontext nicht-funktionaler Laufzeitanforderungen sowohl die für MDD notwendige Domänenkonzepte, wie die Definition von NFE-spezifischen Sprachen und Abbildungsbeschreibungen, als auch Anwendungskonzepte, wie die explizite Spezifikation (Modellierung) der NFE, deren Transformation auf weniger abstrakte Modellierungsschichten sowie die Additive Analyse der NFE. Somit ist es möglich, den PPROKRIS-Ansatz in Verbindung mit MDD einzusetzen, was innerhalb der SuReal-Fallstudie erfolgreich praktisch demonstriert werden konnte.

#### **Bewertung**

Die Diskussion zeigt, dass es prinzipiell möglich ist, den PROKRIS-Ansatz und damit die Anpassung von Vorgehensmodellen an die systematische Unterstützung laufzeitkritischer Anforderungen in gängige Entwicklungsprozessmodelle zu integrieren. Dabei sind verschiedene Abstufungen der Integration möglich. So ist die Integration von Tailoring und

Konkretisierung in Modelle, welche wiederverwendbare Prozessbausteine unterstützen, möglich. Die Integration der Individualisierung ist dagegen abhängig davon, ob die Vorgehensmodelle auf XML-Strukturen abgebildet werden und Konzepte abstrakter Ressourcen wie Rollen und Artefakte beinhalten. Die Unterstützung der individualisierten Vorgehensmodelle durch eine prozessbasierte Ausführungsumgebung ist prinzipiell immer umsetzbar. Grad und Umfang der Formalisierung sind dafür genau abzuwägen (siehe Diskussion in Abschnitt 3.1.2).

### 9.4 Beispielanwendung und Werkzeugunterstützung

Nach der Diskussion der Vollständigkeit der NFE-Prozessmuster und der Anwendbarkeit der PROKRIS-Methodik stellt sich die Frage, ob sich auf der Grundlage der innerhalb der Methodik definierten Konzepte Werkzeuge entwickeln lassen. Die Entwicklung der Werkzeuge innerhalb der Evaluierung des PROKRIS-Ansatzes hat zwei Aufgaben zu erfüllen. Einerseits beweist die Möglichkeit der effizienten Entwicklung von Werkzeugen auf Basis der definierten Metamodellkonzepte die *Machbarkeit* des PROKRIS-Ansatzes. Auf der anderen Seite kann durch die Bereitstellung von Werkzeugen, welche die Funktionalität der Architektur des PROKRIS-Frameworks abdecken, die *Anwendbarkeit* der Methodik gezeigt werden.

Die Werkzeuge an sich werden nicht evaluiert. Bei der prototypischen Implementierung der Werkzeuge wurde vorrangig Wert auf die Umsetzung der Funktionalität gelegt; die Gebrauchstauglichkeit wurde nur am Rand betrachtet. Die Bewertung der Werkzeuge innerhalb einer Nutzerbefragung wird davon mit hoher Wahrscheinlichkeit negativ beeinflusst. Da die PROKRIS-Methodik ein neues Paradigma der Vorgehensmodellerstellung und -anwendung einführt, entsteht mit hoher Wahrscheinlichkeit eine Vermischung der Bewertung der eigentlichen Konzepte der Methodik mit der subjektiven Bewertung der Gebrauchstauglichkeit.

Die Werkzeuge des PROKRIS-Frameworks wurden innerhalb des Projekts SuReal eingesetzt, um eine Beispielanwendung mit Realzeitanforderungen aus dem automotive Bereich zu entwickeln. Dabei konnten Erfahrungen mit der Anwendung der Methodik sowohl bei der Erstellung eines realzeitspezifischen Vorgehensmodells als auch dessen Einsatz bei der Entwicklung der Beispielanwendung unter Nutzung der prozessbasierten Werkzeugumgebung gewonnen werden.

#### 9.4.1 Machbarkeit der technischen Konzepte

Die Architektur des PROKRIS-Framework wird durch verschiedene Werkzeugprototypen umgesetzt. Diese wurden bereits in den vorangegangenen Kapiteln vorgestellt. Dabei wurden die Prototypen entsprechend ihrer Funktionalität innerhalb der Architektur unterschieden. Die entwickelten Prototypen beweisen somit die *Machbarkeit des Ansatzes aus technischer Sicht*.

Sowohl die Anforderungen an die Werkzeugunterstützung als auch die Werkzeugarchitektur selbst wurden bereits in der Arbeit erörtert. Kapitel 3 führte das PROKRIS-Framework als unterstützende Werkzeugumgebung ein und diskutierte verschiedene Ein-



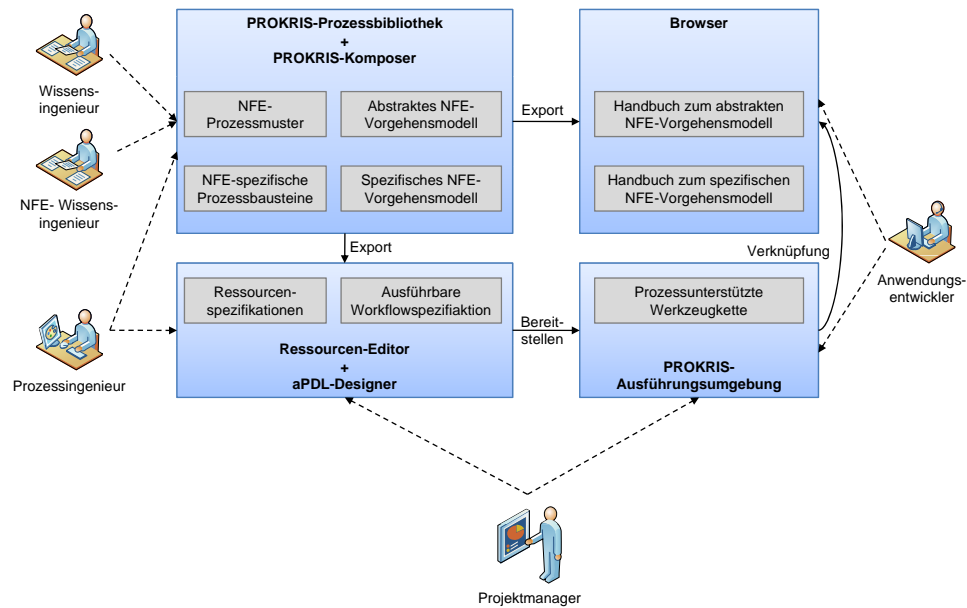


Abbildung 9.1: Werkzeuge und Nutzer des PROKRIS-Frameworks

satzmöglichkeiten aus der Sicht verschiedener Nutzer der Umgebung. In den Kapiteln 6, 7 und 8 wurden architektonische Details zu den einzelnen Komponenten der Architektur besprochen. Daher kann auf dieser Stelle auf eine detaillierte Beschreibung der Architektur verzichtet werden.

## 9.4.2 Anwendung der methodischen Konzepte

Für die Validierung der *Anwendbarkeit der PROKRIS-Methodik* ist außerdem eine Betrachtung der Architektur aus Nutzersicht notwendig. Abbildung 9.1 gibt einen schematischen Überblick über die einzelnen Bestandteile aus der Sicht der verschiedenen Nutzer. Aus Sicht der Anwender der PROKRIS-Methodik können vier Werkzeugkomponenten unterschieden werden. So bildet die PROKRIS-Prozessbibliothek und der -Komposer das erste Werkzeug, welches zur Modellierung, Komposition und Anpassung der statischen Vorgehensmodelle dient. Mit diesem arbeiten sowohl die Wissens- als auch die Prozessingenieure. Zur Erstellung und Verwaltung der Ressourcenspezifikationen und der Individualisierung der ausführbaren Prozessmodelle durch Prozessingenieur oder Projektmanager dienen Ressourceneditor und aPDL-Designer. Diese zweite Werkzeugkomponente ist mit der ersten über eine Exportschnittstelle verbunden, welche die statischen spezifischen NFE-Vorgehensmodelle in ausführbare Prozessmodelle überführt. Die PROKRIS-Ausführungsumgebung als drittes Werkzeug ermöglicht die Orchestrierung der Werkzeuge und Daten, die für die Entwicklung der laufzeitkritischen Software benötigt werden, sowie der involvierten Entwickler. Zusätzlich ist ein Browser zur Anzeige des in die Ausführungsumgebung integrierten, webbasierten Handbuchs notwendig.

### 9.4.3 Beispielanwendung unter Nutzung von PROKRIS-Methodik und -Framework

Das realzeit-spezifische Vorgehensmodell wurde zur Entwicklung einer Beispielanwendung innerhalb des Forschungsprojekts SuReal eingesetzt (SuReal-Vorgehensmodell). Der Einsatz der Werkzeuge des PROKRIS-Frameworks zielt dabei sowohl auf die Entwicklung des Vorgehensmodells als auch auf dessen Anwendung im konkreten Entwicklungsprojekt ab. Es findet also eine Evaluierung der PROKRIS-Methodik sowohl auf der Modellebene M1 als auch auf der Instanz-(Projekt-)ebene M0 statt.

#### Aufgaben der Beispielanwendung

Als Beispiel wurde eine vereinfachte Anwendung aus der Automobilindustrie gewählt. Es umfasst die folgenden Punkte:

- Einhalten eines Mindestabstandes zu einem vorausfahrenden Fahrzeug
- Abstoppen des Fahrzeugs beim plötzlichen Auftreten eines Hindernisses
- Fahren entlang einer vorgegebenen Linie

Dieses vereinfachte Beispiel umfasst Aufgaben eines Tempomats, Bremsassistenten sowie Spurassistenten, wie sie bereits in Mittelklassefahrzeugen im Einsatz sind.

Die Entwicklung erfolgte unter Nutzung eines mit der PROKRIS-Methodik erstellten realzeit-spezifischen Vorgehensmodells. Das zugehörige ausführbare Prozessmodell wurde als prozessbasierte Werkzeugumgebung innerhalb der PROKRIS-Ausführungsumgebung instantiiert und zur Entwicklung eingesetzt. Das Vorgehensmodell und die Werkzeugumgebung integrierte folgende Aktivitäten und entsprechende Werkzeuge:

- Modellierung der Anwendung unter Nutzung des SuReal-UML-Profiles für die Spezifikationen der Zeitbedingungen
- Modellprüfung der zeitspezifischen Aspekte mit dem Werkzeug UPPAAL [UPP]
- Scheduling Analyse mit dem Werkzeug SymTa-S [Sym]
- C-Codeerzeugung
- WCET-Analyse zur Überprüfung der Zeitannahmen mit dem Werkzeug aiT [Abs]
- Bereitstellen auf Lego-Mindstorms-NXT-Robotern<sup>1</sup>

#### Bereitstellung der Vorgehensmodelle für die Beispielanwendung

Die Aufgaben von Tempomat und Bremsassistent implizieren nicht-funktionale Laufzeitanforderungen an die zu erstellende Software. Es handelt sich dabei um harte Realzeitanforderungen, da ein Zusammenstoß der Fahrzeuge unbedingt verhindert werden

---

<sup>1</sup>Lego Mindstorms NXT ist eine Produktserie des Spielwarenherstellers Lego, welche einen programmierbaren Baustein, sowie Elektromotoren, Sensoren und Lego-Technik-Teile (Zahnräder, Achsen, Lochbalken usw.) umfasst, um Roboter und andere autonome und interaktive Systeme zu konstruieren und zu programmieren.

muss. Im Folgenden wird die Anwendung der Methodik zur Erstellung des realzeit-spezifischen Vorgehensmodells und dessen individualisierter Ausführungsspezifikation gezeigt. Zur kontextbasierten Anpassung des Vorgehensmodells wurden die Werkzeuge des PROKRIS-Frameworks eingesetzt. Abbildung 9.2 zeigt die einzelnen Schritte der Methodik anhand entsprechender Ausschnitte aus den in Abbildung 9.1 eingeführten Werkzeugen.

Schritt 1 zeigt die Ablaufstruktur des abstrakten SuReal-NFE-Vorgehensmodells als Ergebnis des Tailoring-Schritts. Zu sehen ist die Integration der Verifikationsschleife in vier verschiedenen Phasen des Entwicklungsprozesses. Schritt 2 stellt den Konkretisierungsschritt, speziell den Einstiegspunkt der Konkretisierung (Menüeintrag **Refinement**), dar. Über diesen gelangt man zum Filteralgorithmus, der die Suche über den Kontext (in diesem Fall der NFE-Kontext) mit den zugehörigen Facetten ermöglicht. Das Werkzeug bietet mögliche NFE-spezifische Bausteine an, aus denen der Prozessingenieur einen passenden auswählen kann (nicht dargestellt). Dieser ersetzt dann das NFE-Prozessmuster. Dieser Konkretisierungsschritt ist iterativ, d.h. es werden alle Muster einzeln durch den Prozessingenieur ersetzt. Im Beispiel wurde die „PIM Verifikation“ durch einen entsprechenden Baustein mit den Aktivitäten „PIM to Timing Analysis Model Transformation“ und „Time related Behaviour Analysis“ ersetzt. Für beide Schritte ist der Prozessingenieur verantwortlich (vergleiche Abbildung 3.5). Er stellt das konkretisierte Vorgehensmodell zur weiteren Nutzung zur Verfügung, indem er sowohl das Handbuch als auch eine ausführbare Repräsentation des Vorgehensmodells exportiert (Schritt 3).

Letztere wird durch den Projektmanager individualisiert, indem konkrete Ressourcen gebunden werden (Schritt 4). Die linke Abbildung zeigt die Aktivitätsknoten der ausführbaren Spezifikation. Die einzelnen Aktivitäten besitzen nun Spezifikationseinträge zur Ressourcenbindung. Auf der rechten Seite ist das Beispiel einer Ressourcendatei im entsprechenden Editor geöffnet. Einzelne Ressourcen oder Gruppen von Ressourcen können an die Aktivitäten gebunden werden. Im Beispiel wird die Gruppe AMEOS (UML-Werkzeug AMEOS mit zugehörigen Transformationswerkzeugen) an die Aktivität PIM Modellierung gebunden. Wurden alle Aktivitäten mit Ressourcen verbunden, kann die ausführbare Beschreibung des SuReal-Vorgehens in der Workflowumgebung bereitgestellt werden (Schritt 5).

Die Grafiken in Schritt 6 liefern eine Idee davon, wie der Entwickler mit der Umgebung arbeitet. Schritt 6.a zeigt die automatisch integrierte Verifikation der PIM-Modelle. Im Beispiel wurde durch die Verifikation mit UPPAAL ein Fehler gefunden. Dem Entwickler werden sowohl weitere Informationen über den Fehler als auch verschiedene Rücksprünge im Vorgehen angeboten (Auflösung des Fehlers im PIM Modell oder neue Aushandlung der Anforderungen - nicht dargestellt). Außerdem kann er während der Bearbeitung immer in das Online-Handbuch wechseln (Schritt 6.b). Er gelangt direkt zu den Hilfeseiten der aktuellen Aktivität. Das Online-Handbuch ist auf Basis der selben Spezifikation wie die ausführbare Beschreibung erstellt und somit konform (dargestellt durch die Pfeile in Schritt 3).

Obwohl die Tätigkeiten des Suchens von Prozessbausteinen (Filtern) und das Komponieren der Modelle nicht dargestellt sind, verdeutlicht das Beispiel die Zusammenhänge der Konzepte der Methodik. Sehr gut sind in der Darstellung die verschiedenen Model-

## 9 Evaluierung von PROKRIS-Methodik und -Framework

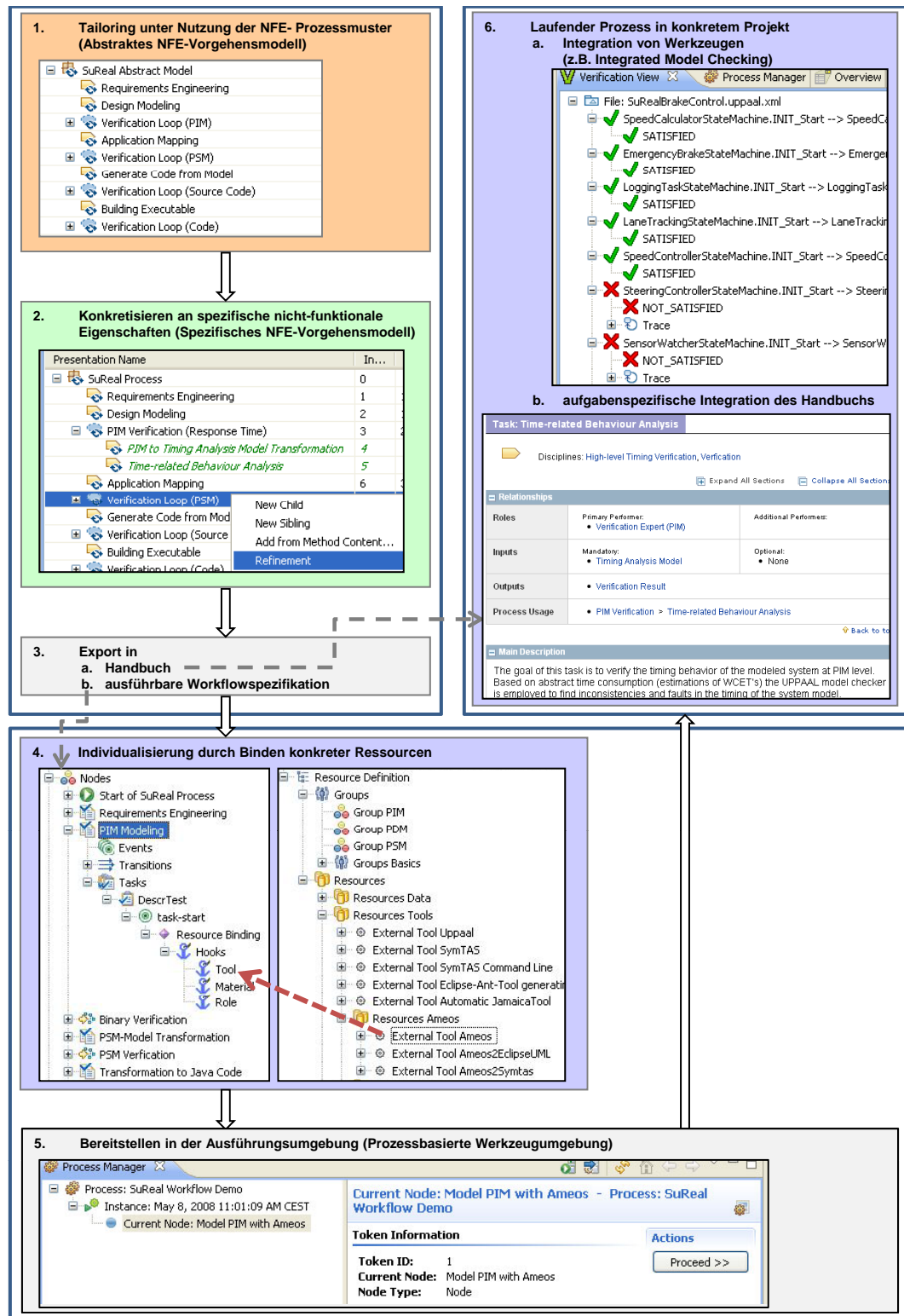


Abbildung 9.2: Anwendung der PROKRIS-Methodik unter Nutzung der Werkzeuge des PROKRIS-Frameworks auf die Vorgehensmodelle des SuReal-Demonstrators

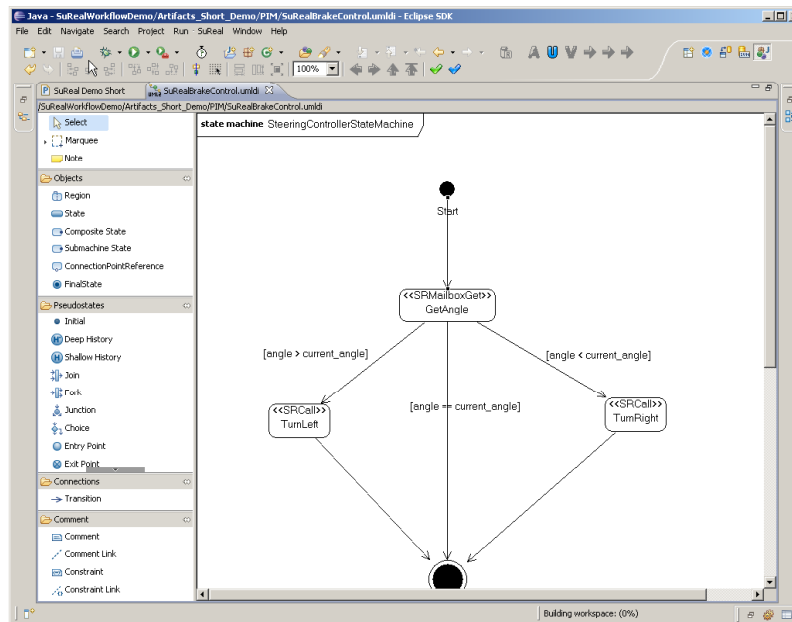


Abbildung 9.3: Modellierung eines UML State Charts für die Beispielanwendung unter Einsatz des SuReal-Realzeit-Profil

lierungsdimensionen der PROKRIS-Methodik zu erkennen. So werden die Vorgehensmodelle vertikal über die drei Abstraktionsstufen verfeinert. Horizontal angeordnet ist die Instanziierungsdimension. Auf der linken Seite befindet sich die Modellebene, wogegen rechts die Instanzen dieser innerhalb der Anwendung in einem konkreten Projekt zu sehen sind.

Die Ablaufstrukturen der Vorgehensmodelle der verschiedenen Abstraktionsstufen können Anhang D entnommen werden.

### Entwicklung der Beispielanwendung unter Nutzung der prozessbasierten Werkzeugumgebung

Das erstellte individualisierte Prozessmodell wurde in die Ausführungsumgebung eingebracht und alle notwendigen Modellierungs- und Verifikationswerkzeuge installiert. Damit stand eine individuell an das Beispielprojekt angepasste prozessbasierte Werkzeugumgebung zur Verfügung, mit der die Beispielanwendung entwickelt werden konnte. Die Abbildungen 9.3 bis 9.6 sollen eine grobe Idee von der Art der Entwicklung mit der PROKRIS-Ausführungsumgebung liefern. Sie zeigen den Ablauf der Aktivitäten des Prozessmusters Verifikationsschleife auf der Ebene der plattformunabhängigen Modellierung (PIM) in vier Schritten. In Abbildung 9.3 ist das UML-Werkzeug TopCased geöffnet in dem der Anwendungsentwickler verschiedene Zustandsmaschinen des Bremssystems modelliert. Zur Modellierung der Zeitbedingung nutzt er das SuReal-UML-Profil. Falls er Erklärungen zu dessen korrekter Anwendung benötigt kann er über die Nutzerschnittstelle der PROKRIS-Ausführungsumgebung (Abbildung 9.4, unten) direkt in die entsprechenden Seiten des Handbuchs wechseln (Abbildung 9.4, oben). Nach der Modellie-

## 9 Evaluierung von PROKRIS-Methodik und -Framework

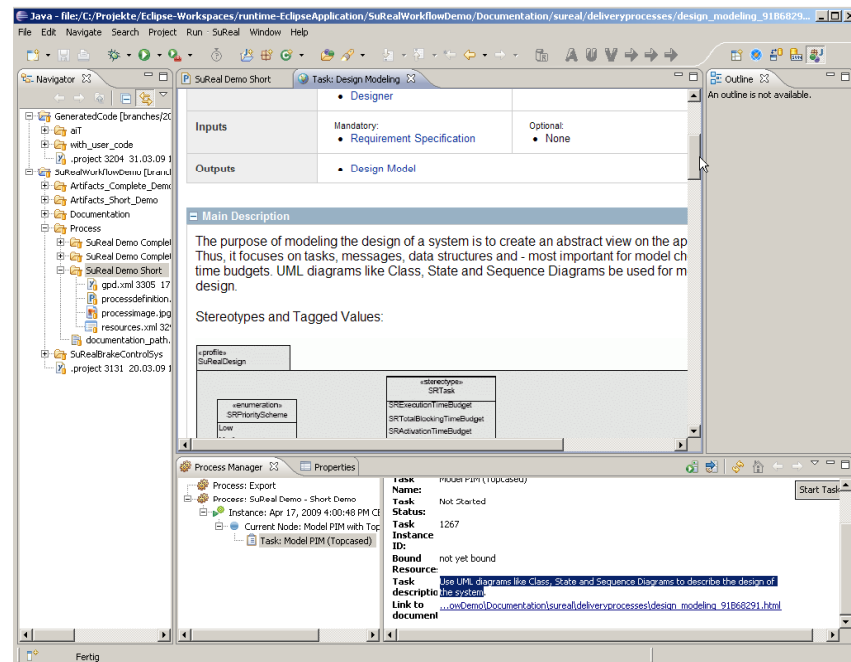


Abbildung 9.4: Anzeige der entsprechenden Seite zur UML Modellierung des integrierten Handbuchs innerhalb der PROKRIS-Ausführungsumgebung

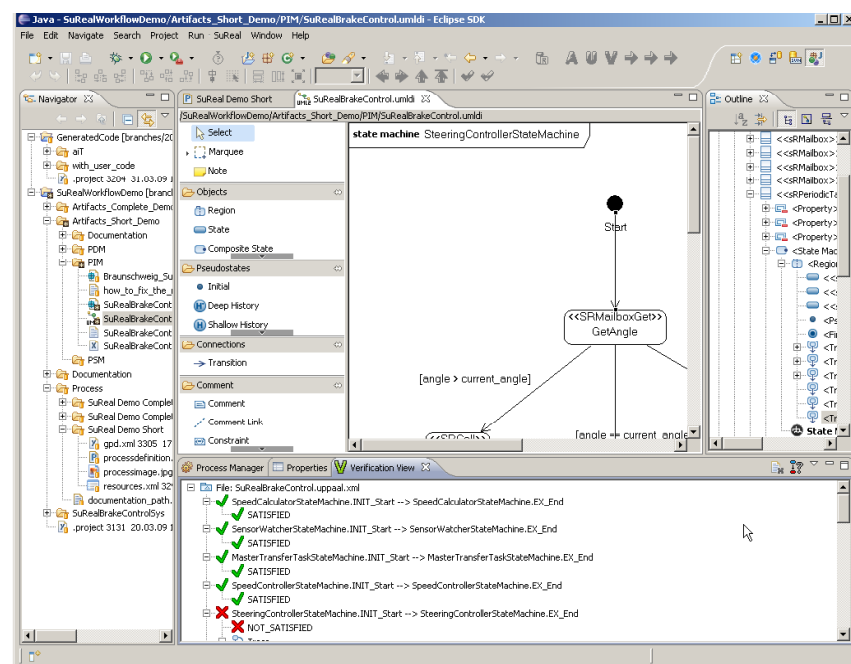


Abbildung 9.5: Integrierter Aufruf und Ausführung der Modellüberprüfung mit UP-PAAL

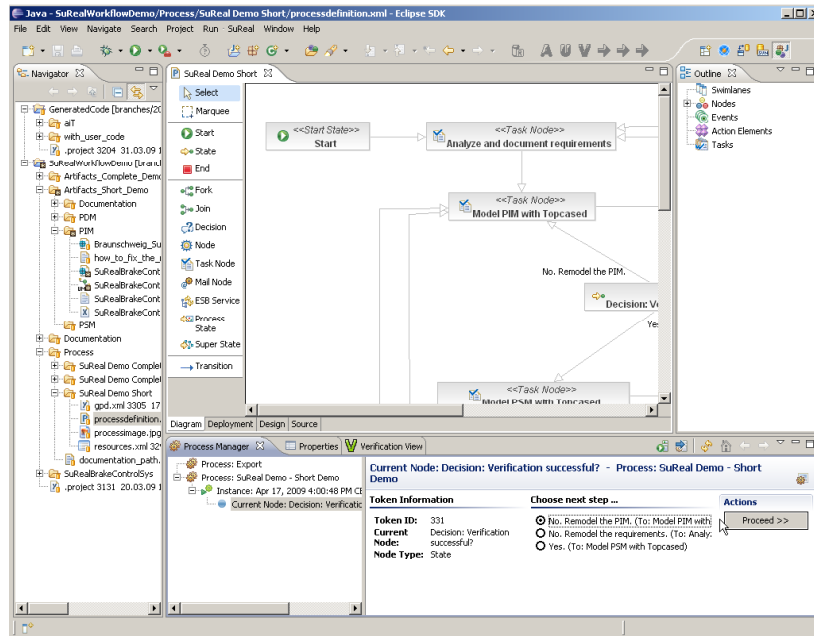


Abbildung 9.6: Entscheidungspunkt innerhalb der Prozessausführung

Die Ausführung des Systems startet der Entwickler die integrierte Modellprüfung. Dabei wird für den Entwickler transparent die Übersetzung in das Prüfmodell und die Verifikation der Zeitanforderungen durchgeführt, da diese Schritte innerhalb der ausführbaren Vorgehensmodellbeschreibung durch automatische Werkzeuge modelliert wurden. Abbildung 9.5 zeigt die Ansicht nach der Verifikation. Es wurden Fehler in der Modellierung gefunden, die dem Entwickler im unteren Fenster angezeigt werden. Bei Fehlern in der Modellüberprüfung muss der Entwickler zurück zur Modellierung gehen. Dies wird ihm durch die Ausführungsumgebung angezeigt. In Abbildung 9.4 ist dieser Entscheidungspunkt in der Abarbeitung zu sehen. Im Fenster rechts unten werden dem Entwickler verschiedene weitere Pfade angezeigt. Er wird in diesem Fall **Remodel the PIM** wählen.

Die Anwendung selbst wurde erfolgreich auf Lego-Mindstorms-NXT-Robotern getestet. Abbildung 9.7 zeigt zwei Roboter, welche während des Tests korrekt reagieren, indem sich beide in Warteposition befinden und nicht weiterfahren, um den Zusammenstoß zu vermeiden<sup>2</sup>. Durch die konsequente Integration und Überprüfung der Zeitanforderungen an das System wurde sichergestellt, dass die Reaktionszeit der Roboter ausreichend kurz ist.

#### 9.4.4 Bewertung

Durch die prototypische Implementierung der Werkzeugunterstützung konnten Erfahrungen mit dem praktischen Einsatz der PROKRIS-Methodik und dem Umgang der

<sup>2</sup>Auf [SuR] sind ausführliche Demos zu Entwicklung und Test der Beispielanwendung bereitgestellt.



Abbildung 9.7: Demonstrator der Beispielanwendung

verwendeten Technologien gesammelt werden. Es wurde demonstriert, dass die Konzepte der PROKRIS-Methodik prinzipiell technisch umsetzbar sind. Außerdem konnte gezeigt werden, dass deren Anwendung unter Nutzung der Werkzeuge des PROKRIS-Frameworks die jeweiligen Nutzer entsprechend ihrer Rollen innerhalb der Vorgehensmodellerstellung und -anpassung sowie die Entwickler bei der Erstellung laufzeitkritischer Systeme angemessen unterstützt. Dies konnte anhand einer Fallstudie innerhalb des SuReal-Projekts erfolgreich demonstriert werden.

Durch den Einsatz der prototypischen Werkzeugunterstützung wurden noch bestehende Herausforderungen aufgedeckt. So setzt die Kategorisierung der wiederverwendbaren NFE-Prozessbausteine mittels der PROKRIS-Facettenklassifikation Disziplin bei der Nutzung voraus. Diese Bedingung müssen alle Bibliotheken und Kataloge erfüllen. An die PROKRIS-Klassifikation wurde auch aus diesem Grunde die Anforderung an eine intuitive und eindeutigen Benutzbarkeit gestellt, was sich bei der Benutzung als gegeben herausgestellt hat. Dies wird vor allem durch die Möglichkeit einer textuellen Beschreibung der einzelnen Kategorien innerhalb der PROKRIS-Prozessbibliothek unterstützt.

Die verwendeten Technologien haben sich als Basis für die Implementierung des PROKRIS-Frameworks bewährt. Durch die Erweiterung bestehender Werkzeuge wurden jedoch im Bereich der Vorgehensmodellkomposition bestehende Konzepte geerbt, welche für die PROKRIS-Methodik überflüssig sind. So muss innerhalb der Bibliothek immer eine *Method Configuration*, also eine eingeschränkte Sicht auf den Inhalt erstellt werden. Dieser Schritt ist redundant zum Einsatz des Kompositionsfilters. Ein Ausschalten der Method Configuration bei der Anwendung der PROKRIS-Methodik war implementierungstechnisch innerhalb des Prototyps nicht in angemessener Zeit möglich, da die entsprechende Funktionalität zentral und mit hoher Kohäsion im Code des EPF Composers verankert ist. Die Existenz des Schrittes hat die Validierung der Anwendbarkeit der Methodik aber nicht negativ beeinflusst.



## 9.5 Zusammenfassende Bewertung der methodischen und technischen Konzepte

Im Folgenden wird die Einschätzung der Ergebnisse der Arbeit in Bezug auf die im einleitenden Kapitel aufgestellten Kriterien für deren Validierung zusammengefasst. Vorab werden kurz die verwendeten wissenschaftlichen Validierungstechniken nach Shaw [Sha02], welche in dieser Arbeit eingesetzt wurden, eingeordnet.

### **Angewendete wissenschaftliche Validierungstechniken**

Im Rahmen der vorliegenden Arbeit wurde die Anwendbarkeit der entwickelten methodischen und technischen Grundlagen durch eine Fallstudie, das „SuReal-Vorgehensmodell“ und dessen Anwendung zur Entwicklung einer Beispielanwendung untersucht. Die Validierung basiert damit hauptsächlich auf einer Evaluation durch Realisierung (Feasibility Study). Damit können die Konzepte der PROKRIS-Methodik an sich validiert werden. Da die innerhalb der Methodik definierten Prozessmuster auf Analysen beruhen, kann deren Gültigkeit und Vollständigkeit nur durch den intensiven Einsatz in realen Projekten bewiesen werden. Durch Beispiele (Examples) und Diskussion (Persuasion) wurde die Verknüpfung mit anderen Entwicklungsprozessmodellen sowie die Erfüllung der notwendigen Eigenschaften eines Vorgehensmodells zur Entwicklung laufzeitkritischer Systeme validiert.

### **Zusammenfassende Bewertung**

*Einheitliche Modellierungssprache:* Eine Forderung an die Methodik war die Bereitstellung einer einheitlichen Modellierungssprache für Prozess- bzw. Vorgehensmodelle auf der Grundlage bekannter Standards. Dazu wurde das SPEM 2.0 Metamodell entsprechend erweitert. Es wurde festgestellt, dass es teilweise inkonsistente Spezifikationen von Konzepten im Metamodell und im Profil gibt. Außerdem besitzt SPEM 2.0 keine Unterstützung für die Prozessausführung. Diese Feststellung deckt sich mit anderen Autoren ([PR09] und [HSGP05]). Die Unterstützung der Prozessausführung erfolgte durch Anbindung der Sprache aPDL, in die Konzepte zur Unterstützung von statischer und dynamischer Ressourcenzuordnung integriert wurden.

*Spezifische Prozessbausteine zur Umsetzung laufzeitbasierter NFE:* Zur Unterstützung der NFE durch Entwicklungsprozesse wurden 13 NFE-Prozessmuster definiert. Diese verallgemeinern Erweiterungen von Softwareentwicklungsprozessen, die sich bei der Umsetzung von laufzeitkritischen NFE als notwendig erwiesen haben. Weiterhin wurde gezeigt, dass diese NFE-Prozessmuster an realzeitspezifische und sicherheitskritische Anwendungsentwicklungen anpassbar sind. Dazu wurden entsprechende NFE-spezifische Prozessbausteine bereitgestellt.

*Durchgängigkeit der Vorgehensmodellierung:* Die Durchgängigkeit vom erweiterten NFE-Vorgehensmodell mit abstrakten NFE-Prozessmustern bis zur Ausführungsunterstützung wurde durch die SuReal-Fallstudie erfolgreich validiert.

*Mögliche Integration mit anderen Softwareentwicklungsprozessen:* Diese Integration ist

möglich und wurde anhand des V-Modells XT, dem RUP und der modellgetriebenen Entwicklung diskutiert.

*Technische Grundlagen und Werkzeugunterstützung:* Das PROKRIS-Framework wurde soweit entwickelt, dass in der Anwendung innerhalb der Fallstudie des SuReal-Projekts sowohl die Durchgängigkeit der Vorgehensmodellerstellung als auch der Einsatz zur Entwicklung einer konkreten Beispielanwendung gezeigt werden konnte. Dazu wurden die entsprechenden Prozessbausteine innerhalb der PROKRIS-Bibliothek modelliert. Außerdem wurde die Verknüpfung von Prozesserstellung und Prozessausführung erfolgreich demonstriert. Um, wie im Kriterienkatalog gefordert, zu gewährleisten, dass das Prozesswissen für die Entwickler präsent ist, wurde das aus dem NFE-spezifischen Vorgehensmodell generierte Handbuch in die prozessbasierte Ausführungsumgebung integriert. Das dieser Ansatz einen sinnvollen Weg zur Bereitstellung von Vorgehensmodellbeschreibungen für Mitarbeiter darstellt, zeigen die Forderungen, die in [Sch09] unter dem Titel „Prozessbeschreibungen, die Mitarbeiter nützlich finden“ aufgestellt werden. Dazu zählt unter anderem:

- keine großen Prozessbücher, sondern die Bereitstellung kleiner fokussierter Einheiten
- die Bereitstellung von Mechanismen, die Informationen zielgerichtet zu finden

Beide Punkte sind im PROKRIS-Framework Realität.

## 10 Verwandte Forschungsarbeiten

In den vergangenen Kapiteln wurde bereits auf verwandte Forschungsarbeiten eingegangen. So wurde in Kapitel 4.1 ein umfassender Überblick über die Unterstützung nicht-funktionaler Systemanforderungen durch Softwareentwicklungsprozesse und -methoden gegeben. Eine Einordnung verschiedener Ansätze zur Klassifizierung nicht-funktionaler Eigenschaften erfolgte in Kapitel 5.5. Außerdem wurden an relevanten Stellen Hinweise auf verwandte Forschungsarbeiten gegeben. In den folgenden Abschnitten werden nun die Beziehungen zu weiteren existierenden Arbeiten erörtert, um die Konzepte des PROKRIS-Ansatzes und damit die Ergebnisse dieser Arbeit entsprechend einzuordnen bzw. abzugrenzen. Die Arbeiten können in die Modellierung von Prozess- und Vorgehensmodellen (also die M1-Ebene) und deren Ausführung (M0-Ebene) sowie Randgebieten eingeordnet werden.

### Modellierung von Prozess- und Vorgehensmodellen

Neben dem in dieser Arbeit verwendeten SPEM 2.0 Metamodell existiert ein weiterer Standard für die Definition eines Metamodells zur Beschreibung von Vorgehensmodellen, das Software Engineering Meta-Model for Development Methodologies (SEMDM ISO 24744, [ISO07]). Es eignet sich grundsätzlich ebenfalls zur Modellierung von Vorgehensmodellen. Es wurde in dieser Arbeit aus zwei Gründen nicht eingesetzt. Erstens ist es nur eingeschränkt erweiterbar, da existierende Elemente weder verändert noch ersetzt werden dürfen [PR09] und zweitens hat SPEM 2.0 den Vorteil der zugehörigen UML-Profil-Definition und der damit verbundenen einfach zu realisierenden Werkzeugunterstützung. [CCCC06] bemängeln, dass SPEM 2.0 nur teilweise semantisch formalisiert ist, was zu den in Kapitel 9 genannten Inkonsistenzen im Standard zwischen Metamodell und Profil beiträgt. Sie definieren daher eine entsprechende Erweiterung auf Basis von OCL. Neben den beiden standardisierten Ansätzen existieren andere nicht standardisierte Arbeiten (siehe [BJF09], [HSGP05] und [GMP<sup>+</sup>03]). Ein Kriterium bei der Entwicklung des PROKRIS-Ansatzes war jedoch der Aufbau auf existierenden Standards.

Sowohl SPEM 2.0 als auch SEMDM definieren keine Ausführungsunterstützung, welche im Kontext des PROKRIS-Ansatzes aber benötigt wurde. Die Entscheidung für die in dieser Arbeit verwendete Sprache jPDL basiert auf einer umfassenden Analyse existierender Ausführungssprachen in [Sch07] und [Mut07]. Rein grafische Modellierungssprachen, wie EPK (Ereignisgesteuerte Prozessketten, [KNS92]) und BPMN (Business Process Modeling Notation, [Obj09]) wurden von vornherein aus der Betrachtung ausgeschlossen. Anhand verschiedener Kriterien werden BPEL (Business Process Execution Language, [IBM03]), XPDL (XML Process Definition Language, [Wor08]), YAWL (Yet Another Workflow Language, [vdAtH05]) und jPDL (jBPM Process Definition Language, [jPD]) bewertet. Tabelle 10.1 fasst die Bewertung der Sprachen anhand funktiona-

Kriterium	BPEL	XPDL	YAWL	jPDL
Spezifikation langfristiger Aufgaben	+	+	+	+
Management von Personalressourcen	o	+	o	o
Erneutes Zuordnen von Personen	o	o	-	o
Zuordnen von Personen zur Laufzeit	o	o	-	o
Formulieren von Bedingungen	-	-	-	-
Reifegrad der Technologie	-	+	-	+
Community und Support	+	o	o	+
Open Source	+	+	+	+
Erweiterbarkeit	+	+	-	+
Standardisierung	-	+	-	-

Tabelle 10.1: Gegenüberstellung der Ausführungssprachen anhand verschiedener Kriterien (nach [Sch07] )

ler (oberer Bereich) und nicht-funktionaler (unterer Bereich) Kriterien zusammen. Sowohl aus funktionaler als auch nicht-funktionaler Sicht stechen XPDL und jPDL hervor. XPDL ist sogar standardisiert, hatte aber zum Zeitpunkt der Technologieentscheidung innerhalb dieser Arbeit aufgrund seiner hohen Komplexität noch Defizite in technischer Unterstützung und Support. Daher wurde jPDL als Ausführungssprache gewählt. Eine wesentlicher Vorteil von jPDL ist die leichte Erweiterbarkeit durch die Implementierung der entsprechenden Workflowumgebung jBPM in Java und der Verfügbarkeit als Open Source und der damit verbundenen Möglichkeit, dass Ressourcenkonzept der PROKRIS-Methodik in eine stabile Workflowumgebung integrieren zu können. Es ist prinzipiell denkbar und auch wünschenswert andere Sprachen und damit Workflowumgebungen als Grundlage der PROKRIS-Ausführungsumgebung einzusetzen. Als guter Überblick über den aktuellen Stand ist [KLL09] zu empfehlen.

Mit der Anbindung einer Ausführungssprache beschäftigen sich auch andere Autoren. Die Ansätze zur Definition der Abbildung sind ähnlich zu dieser Arbeit, wogegen die Zielsprachen verschieden sind. [YLW06] beschreiben beispielsweise eine Abbildung auf XPDL, [BCCG07] und [BSGB07] auf BPEL. Notwendige Konzepte und Definitionen im Forschungsfeld der Verbindung von Softwareentwicklungsprozessen und deren Ausführung beschreiben Feiler und Humphrey schon 1993 ausführlich in [FH93].

Ein interessantes Forschungsgebiet, was parallel zum hier betrachteten *Process Engineering* existiert, ist das innerhalb der Wirtschaftsinformatik vorangetriebene *Method Engineering*. Eine Methode definiert sich hier sowohl aus Prozessaspekten als auch aus Arbeitsproduktbeschreibungen. Das Method Engineering selbst beinhaltet den Entwurf, die Bereitstellung und Adaption von Methoden und entsprechenden Werkzeugen für die Entwicklung von Informationssystemen und wurde erstmals in [Bri96] beschrieben. Das Method Engineering ist innerhalb der Softwaretechnik relativ unbekannt. Henderson-Sellers diskutiert in [HS03] eine Nutzung zur Bereitstellung von wiederverwendbaren Prozess-(Methoden-)wissen und die grundlegende Idee eines Prozessframeworks, wie es auch in dieser Arbeit eingesetzt und erweitert wird (siehe auch Begriffsdiskussion in Kapi-

tel 2.7). Ziel weiterer Forschung sollte es sein, weitere Konzepte des Method Engineerings auf ihren Einsatz in der Softwaretechnik zu untersuchen. Interessant sind hier Ansätze zur Wiederverwendung und Verknüpfung von Methodenfragmenten [RR01] und der Methodenkonstruktion [BWHW05]. [NR08] bietet einen guten Überblick über existierende Umgebungen des computer-gestützten Method Engineering. Weiterverfolgt werden sollte die Doktorarbeit von Johannsen [Joh09], der sich mit der Methodenintegration im Qualitätsmanagement auf der Basis von Method Engineering Konzepten beschäftigen will.

### **Prozessausführung und Ressourcenintegration**

Neben den in der PROKRIS-Ausführungsumgebung eingesetzten Technologien aus dem Bereich der Workflow Management Systeme, kennen auch Plattformen für modellgetriebene Entwicklung, wie openArchitecture Ware (oAW, [oAW]), das Konzept eines Workflows. Dieses hat allerdings ein anderes Ziel als in der Welt der Workflow Management Systeme, an denen sich PROKRIS anlehnt. Ein Workflow bezieht sich dort einzig auf die Definition von Abläufen zur schrittweisen Abarbeitung von Transformationen, ähnlich den Konzepten in *Build*-Werkzeugen (z.B Ant [Apa]). Dabei ist die Interaktion mit Nutzern nicht vorgesehen und beschränkt sich auf das Starten der Transformationen und die Eingabe von Startparametern [Sch07].

Die prozessbasierte Ausführungsumgebung ist als CASE-Umgebung einzuordnen. Unter Computer-Aided Software Engineering (CASE) wird Software verstanden, die benutzt wird, um Softwareprozesse wie Anforderungsanalyse, Entwurf, Programmentwicklung und Tests zu unterstützen. Eine Klassifizierung hilft die verschiedenen Typen von CASE-Werkzeugen und ihre Rolle bei der Unterstützung des Softwareprozesses zu verstehen. Es gibt verschiedene Möglichkeiten der Klassifizierung von CASE-Werkzeugen. So ist es möglich aus der funktionalen Perspektive zu klassifizieren. Dabei kann man z.B. Editoren, Testwerkzeuge, Dokumentationswerkzeuge oder Werkzeuge für das Konfigurationsmanagement unterscheiden. Eine andere denkbare Klassifikation kann aus der Prozessperspektive erfolgen, in der Werkzeuge danach unterteilt werden, welche Prozessphasen bzw. -aktivitäten sie unterstützen. Eine für die Einordnung, der Ergebnisse dieser Arbeit geeignete Klassifikation ist die Einordnung nach der Breite, mit der der Softwareprozess unterstützt wird [Som07]. Danach können Einzelwerkzeuge, Werkzeugsammlungen und CASE-Umgebungen unterschieden werden. Nur letztere unterstützen den ganzen oder zumindest einen Teil des Softwareprozesses. Die Integration verschiedener Werkzeuge und Werkzeugsammlungen innerhalb einer gemeinsamen Umgebung ist das Merkmal dieser Kategorie. Die Standard-Architektur von CASE-Umgebungen basiert auf dem NIST/ECMA-Referenzmodell [CN92]. Es beschreibt drei Formen der technischen Integration von Werkzeugen in eine CASE-Umgebung: Datenintegration, Steuerflussintegration und Präsentationsintegration. Die PROKRIS-Ausführungsumgebung ist darin wie folgt einordbar. Die Datenintegration erfolgt über die Implementierung von Adaptionen zur Transformation von Ausgabedaten eines Werkzeugs in Eingabedaten eines anderen und deren Kopplung über den Steuerflusses. Die Steuerflussintegration ist durch die Workflowausführung der ausführbaren Prozessspezifikation gegeben. Diese unterstützt die Initiierung der einzelnen Werkzeugaufrufe. Die Oberfläche der PROKRIS-

Ausführungsumgebung realisiert die Präsentationsintegration.

Die verschiedenen Arten der Integration werden ähnlich auch in [ABN<sup>+</sup>09] und [Neu02] beschrieben. [Neu02] identifiziert, basierend auf der Arbeit von [TN92], zusätzlich zu den bereits beschriebenen Arten den Zeitpunkt der Integration (a priori, wenn vor der Erstellung des Werkzeugs dessen Integration feststeht und a posteriori, für die Integration von bereits existierenden Werkzeugen) und die Kommunikationsmöglichkeiten (Black Box, Grey Box und White Box Werkzeuge) als Merkmale. [ABN<sup>+</sup>09] vergleicht die Integration von Werkzeugen mit der Orchestrierung von Web-Service-Anwendungen, die vergleichbare Integrationsmöglichkeiten bieten muss. Vertieft wird in dieser Arbeit die modellbasierte Datenintegration am Beispiel der ModelBus-Technologie [HRW09].

Eine Plattform zur Werkzeugintegration stellt jETI (distributed Electronic Tool Integration platform, [Mar05]) dar. Sie bietet eine webbasierte Ausführung einzelner Werkzeuge oder Kombinationen von Werkzeugen an. Diese Funktionalität wird durch Web Services realisiert. Für die Verwendung im PROKRIS-Framework ist dieser Ansatz nicht geeignet, da sämtliche Werkzeuge in Web Services gepackt werden müssten. PROKRIS dagegen sieht Web Services nur als eine Art der Werkzeugintegration neben anderen Möglichkeiten an. Außerdem unterstützt ETI keine Kopplung von Werkzeugen zu Prozessen, die existierende Werkzeugkopplung ist eine Kombination der Funktionalität der Einzelwerkzeuge. Ein Beispiel dafür ist die in [MNS05] beschriebene Kombination von Modelprüfer (Model Checker) und Typprüfer zu einem kombinierten Werkzeug. Die Kombination wird über sogenannte *Tool Coordination Graphs* modelliert und gesteuert.

Neben den Arbeiten zu CASE- und Workflowumgebungen sind weitere Ansätze zu finden, welche sich mit Benutzeroberflächen für die Prozessunterstützung beschäftigen. [KL98] fordert diese unabhängig vom Kontroll- und Datenfluss des zu unterstützenden Prozesses zu gestalten. Bisher stand die Gebrauchstauglichkeit nicht im Fokus der prototypischen Implementierung. Für deren weiteren Ausbau sollte diese Forderung aber berücksichtigt werden. Der *Werkzeug- und Material-Ansatz* beschäftigt sich ebenfalls mit der Benutzeroberfläche. Er stellt das Leitbild des qualifizierten und eigenverantwortlichen Nutzers in den Mittelpunkt, welchem zur Bearbeitung seiner Aufgaben Werkzeuge und Daten (Material) zur Verfügung zu stellen sind [Zül98]. Dieser Ansatz definiert in [RZ95] Ressourcen (Werkzeuge und Material) ähnlich zum PROKRIS-Ansatz mit dem Unterschied, dass nur Werkzeuge für Einzelnutzer betrachtet werden und daher das Konzept der Person nicht notwendig ist. Workflows dagegen orchestrieren auch Personen, weshalb die Behandlung aller drei Ressourcentypen im PROKRIS-Ansatz eine logische Erweiterung darstellt.

Mit dem Bereitstellen von Methodenwissen zur Bearbeitung von Aufgaben bei der Bearbeitung wissensintensiver Prozesse beschäftigt sich [Hol02]. Durch Auswertung der organisatorischen Gegebenheiten (Aufgabe, Rollen) auf der Basis eines Organisationsmodells werden dem Nutzer bei der Abarbeitung einer Aufgabe Listen mit Hinweisen zu entsprechenden Hilfen angeboten. Die entsprechenden Inhalte werden in einer Wissensbasis verwaltet, welche auch die entsprechenden Anfragen bearbeitet. Im Gegensatz zum PROKRIS-Ansatz werden die Inhalte der Wissensbasis nicht dem Prozess an sich zugeordnet, so dass der Nutzer bei Bedarf vom Prozess aus direkt auf das Wissen zugreifen kann. Er muss entweder selbst Anfragen an die Wissensbasis stellen oder es werden ihm

nach Auswertung seiner Arbeit Vorschläge für möglichen Wissensinhalt angeboten.

### **Randgebiete**

Für den weiteren Ausbau der Ausführungsumgebung ist sowohl die Funktionalität von Projektmanagementsystemen als auch der Bereich von CSCW-Systemen Computer (Supported Cooperative Work) von Interesse. Zu beiden Gebieten existiert vielfältige und umfassende Literatur, die im Rahmen der weiteren Arbeit unter den Gesichtspunkten des PROKRIS-Ansatzes analysiert werden sollte.

Für den systematischen Ausbau bietet außerdem die Geschäftsprozessarchitektur „ARIS-House of Business Engineering“ [Sch96] interessante Ansätze. Sie definiert vier Ebenen Prozessoptimierung, Prozessmanagement, Workflow und Anwendung, von denen der PROKRIS-Ansatz bisher die letzten beiden vollständig und die beiden anderen nur zum Teil abdeckt. ARIS beschreibt auf der Ebene der Prozessoptimierung, zusätzlich zu der im PROKRIS-Ansatz unterstützten Modellierung, Aktivitäten zur Analyse, Simulation und Qualitätssicherung von Prozessen. Auf der Ebene des Prozessmanagements fordert ARIS Controlling, Monitoring und Kapazitäts- und Zeitsteuerung, welche durch das PROKRIS-Framework nur ansatzweise umgesetzt werden.





# 11 Zusammenfassung und Ausblick

Das Ziel der Arbeit war die Untersuchung der methodischen Grundlagen der systematischen Prozessunterstützung bei der Entwicklung lauffzeitkritischer Softwaresysteme und die darauf aufbauende technische Umsetzung. In den vorangegangenen Kapiteln wurden die Ergebnisse präsentiert und die PROKRIS-Methodik als eine entsprechende methodische Lösung sowie das unterstützende PROKRIS-Framework als Lösung für die technische Umsetzung vorgestellt. Außerdem wurde der PROKRIS-Ansatzes erfolgreich zur Entwicklung einer Beispielanwendung innerhalb des SuReal-Demonstrators eingesetzt. Die folgenden Abschnitte fassen die Arbeitsergebnisse zusammen und diskutieren die wissenschaftlichen Beiträge der Arbeit und ihre Bedeutung für Softwareentwicklungsprozesse aus softwaretechnischer Sicht. Abschließend werden Ideen für weiterführende Arbeiten diskutiert.

## 11.1 Zusammenfassung

Die Motivation zu dieser Arbeit resultiert aus einer Analyse der Konsequenzen, die sich durch nicht-funktionale lauffzeitkritische Anforderungen für den Softwareentwicklungsprozess ergeben. Diese wurden anhand eines Beispielprojekts aus der Automobilindustrie diskutiert. Dabei zeigte sich, dass die systematische Unterstützung derartiger Eigenschaften sowohl in die grobe Vorgehensplanung als auch in die detaillierte projektspezifische Vorgehensmodellerstellung eingehen muss. Ausgehend vom Beispielprojekt wurden notwendige Eigenschaften eines Frameworks zur systematischen Prozessunterstützung für die Entwicklung lauffzeitkritischer Systeme diskutiert. Dazu zählen die Differenzierung der genannten Ebenen der Vorgehensplanung in Makro- und Mikroprozessmodelle. Außerdem wurden Charakteristika nicht-funktionaler Eigenschaften identifiziert, welche eine, zu funktionalen Eigenschaften abweichende, Behandlung innerhalb der Softwareentwicklung erforderlich machen. Dazu zählen die Optimierbarkeit der Anforderungen, deren eventuelle negative (oder auch positive) Korrelation untereinander, deren Operationalisierung sowie damit zusammenhängend deren Veränderung über den Entwicklungszeitraum. Dabei schränkt sich die vorliegende Arbeit auf die Untersuchung *quantitativer lauffzeitbasierter Eigenschaften* ein. Ein weiterer bei der Entwicklung kritischer Systeme zu beachtender Aspekt ist die Forderung nach fest definierten Vorgehensmodellen, um das Risiko derartiger Projekte überschaubar zu machen. Außerdem erfordert die Komplexität der Modelle und der hohe Anteil an spezifischem Methodenwissen eine sorgfältig abgewogene Ausführungsunterstützung der Entwickler.

Das Fazit ist die Notwendigkeit der Erweiterung von Vorgehensmodellen, um die Unterstützung nicht-funktionaler Eigenschaften zu gewährleisten. Die Erweiterungen wurden in Form von *NFE-Prozessmustern* definiert. NFE-Prozessmuster spezifizieren

Prozessschritte, Methoden, Rollen und Arbeitsergebnisse, die für die Umsetzung nicht-funktionaler Eigenschaften notwendig sind. Sie beschreiben diese generisch bezüglich der konkreten Eigenschaften. Die Spezifikation der NFE-Prozessmuster basiert auf einer umfangreichen Analyse der Unterstützung nicht-funktionaler Laufzeiteigenschaften in existierenden Prozess- und Vorgehensmodellen. Auf der Grundlage dieser Analyse wurden notwendige Eigenschaften des erweiterten Vorgehensmodells definiert. Innerhalb der Anwendungsentwicklung wurden NFE-Prozessmuster für folgende Bereiche eingeführt: Beschreibungstechniken, Kompositionstechniken, Abbildungsmechanismen und Analysetechniken. Zusammen mit der Definition der Einordnung dieser Methoden in ein Vorgehensmodell (ebenfalls über NFE-Prozessmuster) decken die in dieser Arbeit gefundenen Muster alle notwendigen Eigenschaften ab.

Die NFE-Prozessmuster sind als Erweiterungen der Makroprozesse aufzufassen und durch verschiedenste Verfahren und Techniken an die spezifische Ausprägung der nicht-funktionalen Eigenschaften innerhalb eines Entwicklungsprojekts zu instanziierten. Zur Unterstützung dieser Anpassung wurde die *PROKRIS-Methodik der kontextbasierten Anpassung des erweiterten Vorgehensmodells für die Entwicklung laufzeitkritischer Software* entwickelt. Die Methodik unterscheidet drei Stufen von Vorgehensmodellen: abstrakte, NFE-spezifische und ausführbare Modelle. Einer der Vorteile der PROKRIS-Methodik ist das hohe Potenzial der Wiederverwendung von Prozesswissen durch die Verwaltung der NFE-Prozessmuster und deren NFE-spezifischen Umsetzungen in einer Prozessbibliothek. Außerdem erfolgt eine Trennung von abstrakten und konkreten Ressourcenbeschreibungen (Rollen, Werkzeugen und Artefakten) sowie die separate Verwaltung der konkreten Ressourcen einer Organisation innerhalb eines Ressourcenmodells. Die Strukturierung des Inhalts der Prozessbibliothek erfolgt über eine Facettenklassifikation, welche durch den Kompositionsalgorithmus für die Erstellung und Anpassung der verschiedenen Vorgehensmodelle genutzt wird. Orthogonal zur projektspezifischen Anpassung bietet die Methodik eine Ausführungsunterstützung für die Entwickler an.

Zur methodischen Umsetzung des PROKRIS-Ansatzes ist eine einheitliche Prozessmodellierungssprache auf der Metaebene notwendig. Dazu wurde das von der OMG definierte SPEM 2.0 um fehlende Konzepte erweitert. Die Erweiterung umfasst im Wesentlichen die Unterstützung der Vorgehensmodelladaptation durch Wiederverwendung und Verfeinerung, Ausführungssemantik und Ressourcenkonzepte. Außerdem wird die Abbildung auf die ausführbare Prozessbeschreibungssprache aPDL, welche die Workflowsprache jPDL um die Ressourcenbindung zur Laufzeit erweitert, beschrieben.

Die entwickelte Methodik benötigt zu ihrer Anwendung eine entsprechende Werkzeugunterstützung. Bekannte Prozess-Frameworks bieten zwar eine gute Unterstützung für die Verwaltung von Prozessbausteinen, sind aber im Bereich der Prozesskomposition und der Ausführungsunterstützung für den PROKRIS-Ansatz unzureichend. Die Architektur des *PROKRIS-Framework* definiert daher entsprechende Erweiterungen. Außerdem erfolgt die Anbindung an eine *prozessbasierte Ausführungsumgebung*. Diese zeichnet sich dadurch aus, dass sie durch die Laufzeitbindung der konkreten Ressourcen die Instanziierung einer prozessunterstützten Werkzeugumgebung ermöglicht.

Um die Anwendbarkeit der Konzepte zu zeigen, wurde ein Prototyp basierend auf dem Eclipse Process Framework Composer sowie der jBPM Workflowmaschine entwickelt. Die

Vollständigkeit der definierten NFE-Prozessmuster sowie deren NFE-spezifische Anwendung wurde durch zwei Fallstudien, eine im Bereich von Realzeit und eine von Zugriffssicherheit, gezeigt. Das realzeitspezifische Vorgehensmodell wurde erfolgreich zur Entwicklung der Beispielanwendung eines Bremsassistenten innerhalb des Projekts SuReal eingesetzt. Der Demonstrator mit Lego-Mindstorms-Robotern zeigt die Machbarkeit des Ansatzes. Außerdem wurde der Einsatz der PROKRIS-Methodik in Verbindung mit bekannten Vorgehensmodellen, wie V-Modell XT und RUP, diskutiert.

## 11.2 Wesentliche Ergebnisse der Arbeit

Zu Beginn wurden auf der Grundlage der gestellten Ziele dieser Arbeit die dafür erforderlichen wissenschaftlichen Ergebnisse genannt. Im Folgenden wird genauer diskutiert, wie diese Ergebnisse innerhalb des PROKRIS-Ansatzes begründet sind.

Die Hauptergebnisse der Arbeit sind:

1. *Systematisierung der Unterstützung von laufzeitbasierten nicht-funktionalen Eigenschaften durch Entwicklungsprozesse:*

Auf der Basis einer grundlegenden Analyse des aktuellen Stands der Unterstützung nicht-funktionaler Anforderungen in Vorgehens- und Prozessmodellen wurden 13 NFE-Prozessmuster identifiziert und bereitgestellt. Anhand der Fallstudien konnte gezeigt werden, dass beim Einsatz der definierten Muster innerhalb eines Entwicklungsprojekts die kontinuierliche und systematische Umsetzung der nicht-funktionalen Anforderungen gewährleistet ist.

2. *Entwicklung einer Methodik und der zugehörigen Werkzeugunterstützung zur Arbeit mit dem erweiterten Entwicklungsprozess:*

Um sowohl die Wiederverwendbarkeit von Vorgehensmodellen zu ermöglichen, als auch die Erfüllung der nicht-funktionalen Eigenschaften zu gewährleisten, wurde die PROKRIS-Methodik als neues Entwicklungsmodell für die dynamisch-adaptive Bereitstellung der notwendigen Vorgehensmodelle entwickelt. Unterschieden werden die drei Anpassungsstufen abstraktes, NFE-spezifisches und ausführbares Vorgehensmodell. Die Kernidee ist die Wiederverwendung und konsequente Verfeinerung von Vorgehensmodellen über diese Abstraktionsebenen hinweg. Dadurch wird die flexible Erweiterung und Anpassung von Vorgehensmodellen für die systematische Umsetzung nicht-funktionaler Laufzeiteigenschaften möglich. Die Differenzierung des ausführbaren Vorgehensmodells schafft den Projektmanagern die Möglichkeit, bei der Entwicklung der kritischen Teilsysteme eines Softwareproduktes fest definierte Vorgehen im Kleinen festzulegen und zu überwachen, lässt aber andererseits dem Entwickler an unkritischen Stellen genügend Spielraum für die Umsetzung eigener Best Practices.

Zur Umsetzung der Methodik innerhalb einer Werkzeugumgebung wurde die Architektur allgemeiner Prozess-Frameworks um Elemente zur Prozesskomposition und damit verbunden der Modellierung und Filterung der Elemente der NFE-Prozessbausteinlandschaft erweitert.

### 3. *Durchgängigkeit der Prozessmodellierung bis zur Ausführungsunterstützung:*

Die Arbeit stellt außerdem die Verbindung zur Ausführungsebene der NFE-spezifischen Vorgehensmodelle her. Dies ermöglicht die Bereitstellung von prozessunterstützten Werkzeugumgebungen, welche individuell an die Projektbedingungen angepasst sind. Dafür wurde nicht nur die Generierung von ausführbaren Prozessbeschreibungen aus in SPEM 2.0 spezifizierten Prozessmodellen sondern auch die Ressourcenzuordnung zur Laufzeit entwickelt. Dies war notwendig, da die Integration von Rollen und Personen sowie Werkzeugbeschreibungen in der Workflowausführung der gängigen Systeme nicht konsequent umgesetzt ist. Mit der in dieser Arbeit vorgestellten Lösung können die Ressourcen (Rollen, Werkzeuge und Daten) nun zum spätest möglichen Zeitpunkt fest zugeordnet und die Prozessbeschreibung so lange wie möglich diesbezüglich flexibel eingesetzt werden. Ein weiterer Vorteil der Ausführungsumgebung ist die Integration des, aus dem NFE-spezifischen Vorgehensmodell, generierbaren Online-Handbuchs in die prozessgesteuerte Werkzeugumgebung. Damit ist dem Entwickler jederzeit gezieltes Prozesswissen zu den von ihm zu erfüllenden Aufgaben zugänglich.

Im Zusammenhang mit diesen Hauptergebnissen wurden weitere wichtige Ergebnisse erzielt. Dazu zählen:

#### 1. *Evaluierung und Erweiterung des Metamodells SPEM 2.0:*

Das Prozessmetamodell SPEM 2.0 stellte sich beim Einsatz in dieser Arbeit als ausgereift und mächtig dar. Allerdings sind in einigen, für die Unterstützung des PROKRIS-Ansatzes wesentlichen Bereichen Defizite vorhanden, die in der Arbeit durch die Definition eigener Konzepte als Erweiterungen von SPEM 2.0 behoben wurden.

#### 2. *Facettenklassifikation nicht-funktionaler Eigenschaften:*

Im Zusammenhang der Strukturierung der NFE-Prozessbausteinlandschaft mittels einer Facettenklassifikation wurden verschiedene existierende Klassifikationen nicht-funktionaler Eigenschaften analysiert und darauf aufbauend eine Klassifikation quantitativer Laufzeiteigenschaften erstellt, welche in der PROKRIS-Bibliothek angewendet wird. Als hilfreiches Modellierungskriterium dieser Klassifikation stellte sich der facettenbasierte Ansatz nach Glinz heraus, nachdem nicht mehr in funktionale und nicht-funktionale Eigenschaften differenziert wird, sondern alle Eigenschaften nach Art, Repräsentation, Erfüllung und Rolle unterschieden werden. So kann sowohl eine mögliche Veränderung der NFE durch Operationalisierung innerhalb der Facette Art als auch deren verschiedene Verifikation über die Facetten Repräsentation und Erfüllung modelliert werden.

#### 3. *Verbesserung der Standardarchitektur einer Workflowumgebung:*

Die Schnittstelle 3 (Invoked Applications) wurde für den Einsatz der Unterstützung von Softwareentwicklungsprozessen neu konzipiert. Über einen Mediator können verschiedene Werkzeuge aufgerufen werden und Modelle zwischen den Werkzeugen weitergereicht werden. Da die Werkzeugumgebung aus dem jeweiligen NFE-spezifischen Vorgehensmodell entwickelt wird, handelt es sich dabei nicht um starre

Umgebungen (wie den früheren prozessbasierten CASE-Umgebungen) sondern flexibel modellierbaren Werkzeugketten.

4. *Erweiterung der Nutzungsszenarien von Prozess-Frameworks:*

Die Erweiterung der Architektur ist auch für andere Anwendungsfelder einsetzbar. Sowohl der Filteralgorithmus des PROKRIS-Komposers als auch die Übersetzung in die ausführbare Spezifikationssprache sind so entworfen, dass diese unabhängig vom Anwendungskontext der nicht-funktionalen Eigenschaften anwendbar sind.

5. *Domänen-unabhängige Architektur des PROKRIS-Frameworks:*

Die Erweiterungen der Architektur allgemeiner Prozess-Frameworks durch das PROKRIS-Framework sind für andere als das hier betrachtete Anwendungsfeld der Entwicklung von Software mit quantitativen laufezeitkritischen NFE einsetzbar. Sowohl der Kompositionsfilter als auch die prozessbasierte Ausführungsumgebung sowie die Übersetzung in die ausführbaren Spezifikationssprache sind so entworfen, dass diese unabhängig vom Anwendungsfeld verwendet werden können.

## 11.3 Geltungsbereich des Ansatzes

Der vorgestellte Ansatz ist nicht uneingeschränkt auf alle Entwicklungsprojekte anwendbar. Eine erste Einschränkung liegt der Arbeit mit der Beschränkung auf quantitative (also messbare) nicht-funktionale Laufzeiteigenschaften zu Grunde. Eine weitere ist durch den Overhead der Prozessmodellierung gegeben. Einerseits muss die Prozessbibliothek sorgfältig aufgebaut und gewartet werden. Andererseits werden mehrere Modelle und damit Dokumente verschiedener Ausbaustufen von Vorgehensmodellen erzeugt. Diese müssen ebenfalls verwaltet und gepflegt werden. Daher ist die Nutzung eines derartig umfangreichen und damit in seiner Funktionalität mächtigen Ansatzes nur bei großen Projekten angezeigt.

Die in dieser Arbeit entwickelten Prozessmuster und NFE-spezifischen Prozessbausteine beruhen auf einer umfassenden Analyse des aktuellen Stands der Technik im Bereich der Unterstützung von nicht-funktionalen Eigenschaften durch bekannte Vorgehens- und Prozessmodelle. Deren Evaluierung erfolgte durch Abbildung auf die Vorgehensmodelle zweier Fallstudien sowie den Einsatz zur Entwicklung einer Beispielanwendung. Zur systematische Verbesserung des Inhalts der PROKRIS-Prozessbibliothek als Basis des PROKRIS-Ansatzes durch Prozessevaluierungsverfahren ist eine Anwendung in weiteren konkreten Entwicklungsprojekten notwendig.

Eine mögliche Hürde bei der Anwendung könnte es sein, dass die PROKRIS-Methodik ein neues mentales Modell des Prozessingenieurs von Vorgehensmodellen und deren Erstellung vom „Kleinen zum Großen“, im Gegensatz zum bisher üblichen „vom Großen zum Kleinen“ (Zuschneiden (Tailoring) von Modellen) notwendig macht. Die PROKRIS-Methodik wurde zwar durch die prototypische Werkzeugunterstützung und deren Anwendung zur Entwicklung einer Beispielanwendung innerhalb der SuReal-Fallstudie evaluiert. Um die Akzeptanz der Nutzer bewerten zu können, sind umfangreichere Projekterfahrungen beim Umgang mit der Methodik durch Nutzerbefragungen auszuwerten.

## 11.4 Weiterführende Arbeiten

Aufbauend auf den Ergebnissen können verschiedene weitere interessante Forschungsfragen identifiziert werden. Zudem wurden während der Arbeit neue Fragen aufgeworfen, deren ausreichende Untersuchung den Rahmen der Arbeit gesprengt hätten. Die weiterführenden Arbeiten lassen sich in drei Kategorien einteilen: Ausbau und Verbesserung des PROKRIS-Ansatzes, Untersuchung des Einsatzes in anderen Anwendungsdomänen sowie Anwendung der MDA für die Entwicklung von Softwareentwicklungsprozessen.

### **Ausbau und Verbesserung der PROKRIS-Methodik und des -Frameworks**

Die definierten NFE-Prozessmuster beinhalten Methoden zur systematischen Umsetzung nicht-funktionaler Eigenschaften während des Kernprozesses der Softwareentwicklung. Aber auch in den unterstützenden Prozessen ist es möglich und notwendig, entsprechende NFE-Prozessmuster zu definieren. Beispiele für Bereiche, welche zur Definition von Methodeninhalten für die PROKRIS-Prozessbibliothek analysiert werden sollten, sind das Risikomanagement sowie Kostenanalysen und -abschätzungen im Zusammenhang mit nicht-funktionalen Anforderungen. Zu Letzterem gehört die Untersuchung von Metriken für nicht-funktionale Eigenschaften.

Die Definition des erweiterten Vorgehensmodells ermöglicht es, Defizite in der Bereitstellung und Entwicklung von konkreten Verfahren bzw. Techniken zur Umsetzung anderer nicht-funktionaler Eigenschaften aufzuzeigen. Die Durchführung dieser Analyse (Kapitel 4) schafft eine strukturierte Ausgangsbasis für die Definition weiterer Forschungsarbeiten zur Beseitigung derartiger Lücken in der Unterstützung.

Bisher beschränkt sich der Ansatz auf quantitative Laufzeiteigenschaften. Eine weitere interessante Forschungsfrage ist es daher, zu untersuchen inwieweit mit diesem Ansatz auch qualitative Laufzeiteigenschaften, wie Benutzbarkeit (Usability) oder sogar Entwicklungszeiteigenschaften unterstützbar sind.

Die Werkzeugunterstützung durch das PROKRIS-Framework ist bisher nur prototypisch realisiert, da innerhalb der Arbeit die Machbarkeitsanalyse der Methodik das Ziel war. Neben der vollständigen Implementierung sind weitere konzeptionelle Arbeiten in der Werkzeugunterstützung denkbar. Dazu zählt die Verbesserung der intuitiven Nutzbarkeit der Werkzeuge. Dabei spielt die Evaluierung des bereits erwähnten neuen mentalen Modells der Prozesserstellung eine Rolle. Weitere offene Fragen beschäftigen sich mit der Ausführungsunterstützung. Dazu zählen die Umsetzung der Verbindung von Architekturentscheidungen und dem ausführbaren Prozessmodell. Bei Aufsplittung von Komponenten muss eine dynamische Erzeugung, Verarbeitung und Verwaltung entsprechender Subprozesse erfolgen. Die Problematik und entsprechende Lösungsansätze werden in Abschnitt 5.9 dieser Arbeit diskutiert. Die Architekturen der untersuchten Workflowmaschinen unterstützen nur statisch vordefinierte Subprozesse. Es sind also sowohl neue Sprachkonzepte als auch Erweiterungen der Architektur von Workflowmaschinen notwendig. Ein weiterer Punkt ist die Untersuchung der Unterstützung der Kommunikation der nicht-funktionalen Anforderungen innerhalb der prozessbasierten Ausführungsumgebung. Dazu gehört einerseits die für den Entwickler sichtbare Rückverfolgbarkeit nicht-funktionaler Eigenschaften. Andererseits müssen die nicht-

funktionalen Anforderungen auch in die umgekehrte Richtung kommuniziert werden, so dass jeder Entwickler die Anforderungen an das von ihm bearbeitete Subsystem und, wenn möglich, der unmittelbar damit kommunizierenden Subsysteme kennt. Ein Lösungsansatz dafür ist die Unterstützung von Vertragskonzepten durch die Prozessausführungsumgebung. In diesem Zusammenhang ist es außerdem interessant zu untersuchen, inwieweit der Erfüllungsgrad der nicht-funktionalen Anforderungen auswertbar ist und welchen Einfluss die Weitergabe dieses Wissens an die Entwickler auf den Erfolg von Projekten hat.

Sowohl die PROKRIS-Methodik als auch das -Framework sind unter dem Aspekt entworfen, dass sie für das kooperative Arbeiten in verteilten Umgebungen einsetzbar sind. Allerdings ist die notwendige Unterstützung innerhalb der Ausführungsumgebung noch unzureichend untersucht und daher nicht implementiert. In [Sch07] wurden bereits Konzepte und Prototypen für das Arbeiten mit verschiedenen Teams entwickelt. Diese sind in die Werkzeugunterstützung zu integrieren. Noch unzureichend untersucht ist die Einbindung von Werkzeugen und Daten in verteilten Umgebungen. Ein möglicher Ansatz ist die Erweiterung des in dieser Arbeit entwickelten Ressourcenmodells zu einem Metamodell für die generische Modellierung von Daten und deren Transformationen, um eine datenbasierte Werkzeuganbindung realisieren zu können.

Die Formalisierung des Softwareentwicklungsprozesses ermöglicht außerdem die Integration von Metriken und damit einhergehend die Verbesserung der Prozesse durch Adaption. Diese Form der Qualitätssicherung sollte ebenfalls in weiteren Forschungsarbeiten untersucht werden.

### **Einsatz in anderen Anwendungsdomänen**

Die Motivation für diese Arbeit war die Unterstützung von Stakeholdern bei der Arbeit mit Vorgehensmodellen für laufzeit-kritische Anwendungen. Dabei handelt es sich um komplexe Prozessmodelle, welche außerdem eine formale Definition erfordern. Obwohl der entwickelte Ansatz bezüglich des Inhalts der PROKRIS-Bibliothek NFE-spezifisch ist, sind andere Konzepte generisch einsetzbar. Interessant ist es daher, das Potential der Nutzung für andere komplexe Vorgehensmodelle bzw. Prozesse zu analysieren. Ansatzpunkte sind dabei der PROKRIS-Komposer und damit die Anpassung von Prozessen vom abstrakten Vorgehensmodell auf der Makroebene bis hin zum projektspezifischen ausführbaren Prozessmodell. Ein anderer Ansatzpunkt ist die Ausführungsumgebung. Das PROKRIS-Framework unterstützt eine workflowbasierte Abarbeitung, da diese für die laufzeitkritischen Systeme von Vorteil ist. Für den Einsatz in anderen Anwendungsdomänen ist zu untersuchen, ob die definierten Konzepte auch auf eine weniger restriktive Abarbeitung, basierend auf Aufgabenlisten und Rollenzuordnungen (wie z.B. Trac [Edg]), abbildbar sind.

### **MDA für die Entwicklung von Softwareentwicklungsprozessen**

Mit dem PROKRIS-Ansatz ist es möglich Familien von Entwicklungsprozessen zu behandeln. Osterweil hat bereits 1987 in seinem Artikel „Software Processes are Software too“ [Ost87] geschrieben, dass man die Entwicklung und Ausführung von Softwareprozessen mit der Modellierung und Implementierung von Software selbst vergleichen kann

und demzufolge auch die Methoden und Techniken der Softwareentwicklung auf die Entwicklung von Softwareprozessen anwendbar sind. Dies ist auch immer wieder deutlich in dieser Arbeit zu erkennen. Schon Abbildung 3.1 zum Prozesslebenszyklus in Kapitel 3 erinnert stark an den Softwareprozess selbst. Daher ist es naheliegend, darüber nachzudenken, Techniken die MDA und MDD bieten, für die Entwicklung von Prozessen einzusetzen und zu untersuchen, ob der PROKRIS-Ansatz auf die Entwicklung von Produktlinien für Prozesse erweiterbar ist.

Es existieren bereits einige interessante Ansätze, welche sich mit Produktlinien für Prozesse auf der Basis der MDA beschäftigen ([SCLJO07], [BB01]). Weiterhin stellt [Mül07] eine neue Entwicklungsmethode die *Process Driven Architecture* (PDA) zur Entwicklung und Bereitstellung von prozessbasierten Werkzeugunterstützungen für wissen-intensive medizinische Forschungsprozesse ein. Dazu werden sowohl Konzepte der MDA als auch des Method Engineering eingesetzt. Die Ansätze sollten in Verbindung mit dem PROKRIS-Ansatz untersucht werden. Allerdings beschäftigen sich diese Arbeiten mit Geschäftsprozessen wogegen der Anwendungsbereich des PROKRIS-Ansatz Softwareentwicklungsprozesse sind. In die Erstellung dieser geht Erfahrungswissen von Prozessingenieuren und Projektmanagern in Entscheidungen innerhalb der Komposition und projektspezifischen Konfiguration der Vorgehensmodelle bzw. die Auswahl von Prozessbausteinen ein. Daher ist zu untersuchen, wie dieses implizite Wissen modellierbar und automatisiert einsetzbar ist.

Zusammenfassend lässt sich sagen, dass die Entwicklung von PROKRIS-Methodik und -Framework einen wesentlichen Beitrag zur systematischen und wiederverwendbaren Modellierung sowie der Ausführung von Softwareentwicklungsprozessen nicht nur im Bereich der Entwicklung laufzeitkritischer Systeme liefert. Darüberhinaus leistet die Arbeit einen Beitrag zur Anwendung der Regeln der Softwaretechnik auf die Entwicklungsprozesse selbst. Weiterhin wird durch die prozessbasierte Ausführungsumgebung auf der Grundlage der NFE-spezifischen Vorgehensmodelle kooperatives Arbeiten und nicht zuletzt eine Dokumentierbarkeit der Entwicklung möglich. Dies hat direkte Auswirkung auf die erreichbare CMMI-Stufe sowie die Zertifizierung der mit der Umgebung erstellten Software. Diese Arbeit bildet eine konzeptionelle und methodische Basis für eine Vielzahl weiterer Forschungsfragen.



# A Katalog der Prozessmuster

Im Folgenden werden die NFE-Prozessmuster detailliert, anhand der in Tabelle 4.2 aufgestellten Vorlage, beschrieben.

Nr.	Prozessmuster	Klassifikation
A.1	Explizite Spezifikation der NFE	Aktivitätsmuster, Anwendungsentwicklung
A.2	Identifikation der kritischen Szenarien	Aktivitätsmuster, Anwendungsentwicklung
A.3	Konfliktanalyse	Aktivitätsmuster, Anwendungsentwicklung
A.4	Verifikationsschleife	Aktivitätsmuster, Anwendungsentwicklung
A.5	Additive Verifikation der NFE	Aktivitätsmuster, Anwendungsentwicklung
A.6	Transformation der NFE-Spezifikationen	Aktivitätsmuster, Anwendungsentwicklung
A.7	Analyse der Operationalisierung	Aktivitätsmuster, Anwendungsentwicklung
A.8	Dekomposition der NFE	Aktivitätsmuster, Anwendungsentwicklung
A.9	Komposition der NFE	Aktivitätsmuster, Anwendungsentwicklung
A.10	Entwicklung entlang des kritischen Pfades	Phasenmuster, Anwendungsentwicklung
A.11	Inkrementelle Entwicklung der kritischen Systembereiche	Phasenmuster, Anwendungsentwicklung
A.12	NFE-spezifische Sprache	Aktivitätsmuster, Domänenentwicklung
A.13	NFE-spezifische Abbildungsbeschreibung	Aktivitätsmuster, Domänenentwicklung

Tabelle A.1: Übersicht über die definierten NFE-Prozessmuster

Tabelle A.1 gibt einen Überblick über die definierten Muster. Es sind nur Prozessmuster für Aktivitäten bzw. Phasen aufgeführt. Die einzelne Aktivitäten innerhalb der Musterbeschreibungen können wiederum als Prozessmuster für Aufgaben (Task Process Pattern) verallgemeinert werden. Dies würde den Umfang der Arbeit übersteigen. Im

Prototyp der PROKRIS-Prozessbibliothek sind diese Muster entsprechend modelliert.

## A.1 Explizite Spezifikation der NFE

### Name und Klassifikation

Explizite Spezifikation der nicht-funktionalen Eigenschaften(NFE) (Explicit Specification of Non-functional Requirements (NFR))

Aktivitätsmuster, Prozessmuster der Anwendungsentwicklung

### Kurzbeschreibung

Das Muster beschreibt die notwendigen Aktivitäten und Artefakte zur Integration der expliziten Spezifikation nicht-funktionaler Eigenschaften in der Systemmodellierung.

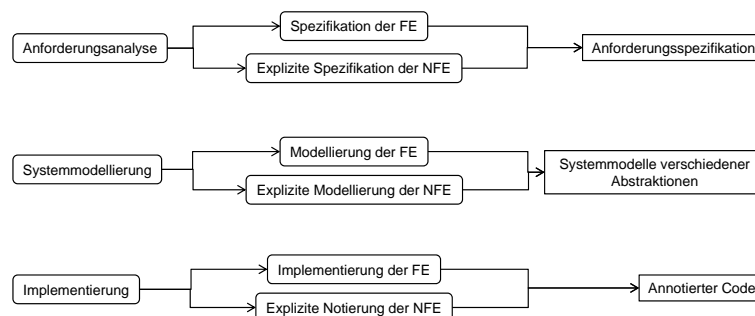
### Problem

Nicht-funktionale Anforderungen bzw. Eigenschaften müssen innerhalb des Systemmodells explizit spezifiziert werden. Das bedeutet, dass die Eigenschaften mit konkreten messbaren Werten zu belegen sind. Grund ist einerseits, dass nur so die Überprüfung der Erfüllung der Eigenschaften und damit die Qualitätssicherung möglich ist. Andererseits ist die Erfüllung derartiger Eigenschaften optimierbar. Durch die Angabe konkreter Maße sind Kosten-/ Nutzenanalysen möglich.

### Lösung

Dem Entwickler werden die notwendigen Beschreibungsmittel zur Modellierung nicht-funktionaler Eigenschaften bereitgestellt. Es wird eine Aktivität in den Prozess integriert, welche die explizite Spezifikation der nicht-funktionalen Eigenschaften als messbaren Wert fordert. Die Modellierung hat für Nutzer-, Anwendungs- und Plattform(Laufzeit)-Eigenschaften zu erfolgen. Daher muss die Aktivität in allen Phasen des Prozesses eingebunden werden, in denen Systeme modelliert oder spezifiziert werden. Die Notwendigkeit der expliziten Spezifikation der nicht-funktionalen Eigenschaften ist den Entwicklern innerhalb der Werkzeuge präsent zu machen.

### Struktur



### Aktivitäten

Das Muster beinhaltet nur eine Aktivität, welche je nach Entwicklungsphase unterschiedliche Spezifikationsmittel verwendet.

*Explizite Spezifikation der NFE:* Nicht-funktionale Anforderungen bzw. Eigenschaften sind unter Nutzung der bereitgestellten Beschreibungsmittel explizit zu spezifizieren. Dabei kann diese Aktivität parallel zur funktionalen Spezifikation erfolgen. Bedingung ist, dass die Ergebnisse beider Aktivitäten in ein gemeinsames Modell des Systems einfließen. Innerhalb der Anforderungsentwicklung werden die Anforderungen in geeigneter Weise spezifiziert, in der Modellierungsphase modelliert und im Codemodell durch Annotationen notiert.

### Produktartefakte

*Input:* NFE-spezifische Beschreibungsmittel, funktionale Modelle

*Output:* integriertes Modell der funktionalen und nicht-funktionalen Aspekte einer Anwendung, eines Subsystems oder einer Komponente

### Kontext

Durch die in die Anwendungsentwicklung involvierten Rollen erfolgt die explizite Spezifikation der NFE. Dazu gehören sowohl Anforderungsanalysten, als auch Entwickler, Tester und Verifikationsexperten. Ziel ist es, die innerhalb der Anforderungsanalyse extrahierten subjektiven NFE auf konkrete Maßangaben herunterzubrechen. Die Angabe konkreter Maße für die laufezeitkritischen nicht-funktionalen Eigenschaften muss in allen Abstraktionsgraden der Systemmodellierung vorliegen, da nur so deren durchgängige Behandlung von der Anforderungsanalyse bis zur Ausführung zu gewährleisten ist.

Dem Festlegen der konkreten Maße ist eine Analyse der Anforderungen mit integrierter Analyse möglicher Konflikte (Prozessmuster „Konfliktanalyse“) voranzustellen. Um die Konsistenz der Spezifikationen zu gewährleisten, sind außerdem Abbildungsregeln zwischen den Abstraktionsebenen zu definieren (Prozessmuster „Transformation der NFE-Spezifikation“). Die damit unterstützte automatische Transformation kann in der Regel nicht alle Lücken zwischen den Modellen schließen [GPR06], so dass eine manuelle Verfeinerung durch die beteiligten Rollen notwendig ist.

Die explizite Spezifikation der NFE sollte durch die verwendeten Modellierungswerkzeuge unterstützt werden.

### In Beziehung stehende Muster

*NFE-spezifische Sprache:* Die notwendigen Sprachmittel zur expliziten Beschreibung nicht-funktionaler Eigenschaften durch Maße stellt der NFE-Wissensingenieur zur Verfügung. Die dafür notwendigen Aktivitäten sind in diesem Prozessmuster beschrieben.

*Alle anderen NFE-Prozessmuster der Anwendungsentwicklung:* Die explizite Spezifikation der nicht-funktionalen Eigenschaften ist grundlegende Voraussetzung für die Anwendung aller anderen Prozessmuster der Anwendungsentwicklung, da diese mit den NFE-Spezifikationen arbeiten.

## A.2 Identifikation der kritischen Szenarien

### Name und Klassifikation

Identifikation der kritischen Szenarien (Identification of critical Use Cases)  
Aktivitätsmuster, Prozessmuster der Anwendungsentwicklung

### Kurzbeschreibung

Das Muster beschreibt die notwendigen Prozessschritte zur Identifikation der kritischen Szenarien (Use Cases). Diese sind Voraussetzung, um die iterative Entwicklung entlang des kritischen Pfades umsetzen zu können.

### Problem

Ein Ansatz zur effizienten und risikoarmen Entwicklung laufzeitkritischer Systeme beruht auf der inkrementellen und iterativen Entwicklung des Softwaresystems entlang des kritischen Pfades. Der kritische Pfad beschreibt die Verkettung derjenigen Systemfunktionen, die untrennbar mit der Erfüllung der geforderten nicht-funktionalen Eigenschaften zusammenhängen. Im Falle von Zeitanforderungen ist es die Kette von Funktionsaufrufen, welche in der Summe die längste Dauer aufweist. Die Analyse der kritischen Szenarien eines Systems ist eine Voraussetzung für das Festlegen des kritischen Pfades.

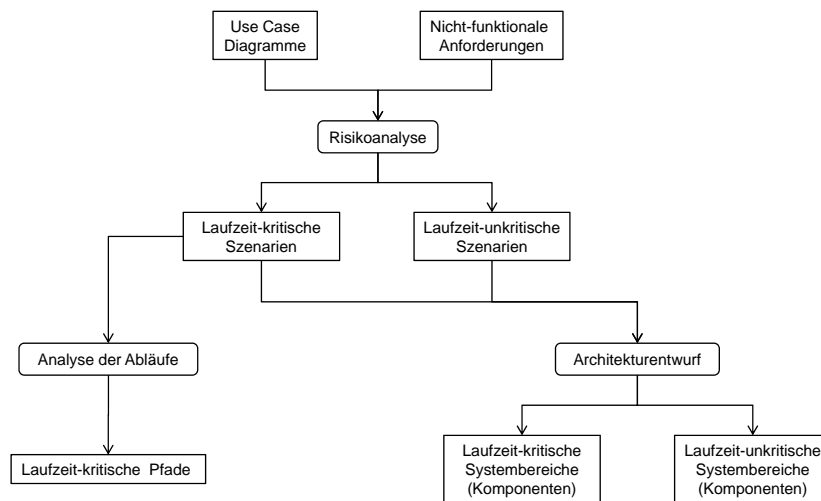
### Lösung

Kritische Szenarien sind die Nutzungsszenarien eines Softwaresystems, welche entweder für die Erfüllung nicht-funktionaler Anforderungen an eine Systemfunktionalität oder für die Einhaltung von absoluten (also nicht-verhandelbaren) nicht-funktionalen Nutzerkriterien ausschlaggebend sind. Die Auswahl der kritischen Szenarien ist demnach immer mit einer Risikoanalyse verbunden. Als kritisch werden dabei solche Szenarien eingestuft, welche das Risiko besitzen, dass bei der Nichterfüllung der nicht-funktionalen Anforderungen das System fehlerhaft reagiert bzw. die Nutzerakzeptanz nicht mehr gewährleistet ist. Die kritischen Nutzerszenarien werden durch Use Case Modelle dargestellt.

Wie bereits im Abschnitt Problem geschildert, basiert die Entwicklung laufzeitkritischer Systeme auf der Entwicklung entlang des kritischen Pfades. Dabei sind nicht alle möglichen Abläufe innerhalb eines Szenarios als kritisch einzustufen. Der kritische Pfad (*key szenario*) ist daher nur eine mögliche Sequenz eines Kontroll- oder Datenflusses, welche in einem zusätzlichen Analyseschritt extrahiert werden muss.

Auf der Grundlage der Analyse der kritischen Szenarien ist es außerdem möglich, während des Architekturentwurfs, in Bezug auf nicht-funktionale Laufzeiteigenschaften, kritische und unkritische Systembereiche bzw. Komponenten zu differenzieren. Diese Trennung ist nicht nur als Managementunterstützung hilfreich, sondern ermöglicht die systematische Entwicklung unter Nutzung der iterativen und inkrementellen Entwicklung.

## Struktur



## Aktivitäten

*Risikoanalyse:* Die Risikoanalyse verfolgt das Ziel, mögliche Risiken frühzeitig zu erkennen, zu bewerten und mögliche Gegenmaßnahmen zu planen [LL07]. Im Kontext der nicht-funktionalen Anforderungen dient die Risikoanalyse dazu, die in diesem Sinne kritischen Anwendungsszenarien zu identifizieren.

*Analyse der Abläufe:* Um den kritischen Pfad eines Systems zu finden, ist eine Analyse der Szenarien unter Verwendung dynamischer Modelle notwendig. Dabei können sich sowohl Kontroll- als auch Datenflüsse als kritisch herauskristallisieren. Dies wird im Wesentlichen von der Art der geforderten NFE beeinflusst.

*Architekturentwurf:* Die Differenzierung von laufzeit-kritischen und unkritischen Szenarien sollte in Modellierungsentscheidungen während des Architekturentwurfs einfließen. Eine möglichst saubere Trennung dieser Szenarien im Architekturmodell und damit verbunden der expliziten Spezifikation der NFE an den Schnittstellen ermöglicht eine Parallelisierung der Entwicklung von Teilsystemen. Dazu ist die Definition unabhängiger Schnittstellen notwendig, deren nicht-funktionale Eigenschaften sich gegenseitig nicht beeinflussen.

## Produktartefakte

*Input:* Dokumente der Anforderungsanalyse, insbesondere Use Case Diagramme und nicht-funktionale Anforderungen

*Output:* kritische Szenarien, kritischer Pfad des Systems, kritische Architekturkomponenten

### Kontext

Während die Identifikation der kritischen Szenarien in den Verantwortungsbereich der Anforderungsanalysten fällt, sind die Analyse des kritischen Pfades und die Entscheidungen bezüglich der Systemarchitektur nur in Zusammenarbeit mit Architekten und Verifikationsexperten möglich. Dabei ist nur ein Teil der Szenarien als risikobehaftet einzustufen. Deren Ermittlung sollte in enger Verzahnung mit einer Risikoanalyse einhergehen.

### In Beziehung stehende Muster

*Entwicklung entlang des kritischen Pfades:* Dieses Muster setzt die Definition des kritischen Pfades auf Basis der Analyse der kritischen Szenarien voraus.

*Inkrementelle Entwicklung der kritischen Systembereiche:* Die Differenzierung der kritischen und unkritischen Systembereiche ist eine Voraussetzung für die Anwendung dieses Musters.

## A.3 Konfliktanalyse

### Name und Klassifikation

Konfliktanalyse (Analysis of Non-functional Conflicts)

Aktivitätsmuster, Prozessmuster der Anwendungsentwicklung

### Kurzbeschreibung

Das Muster führt eine Aktivität „Konfliktanalyse“ in die Systementwicklung ein, in welcher Konflikte von nicht-funktionalen Anforderungen aufgedeckt und gelöst werden sollen.

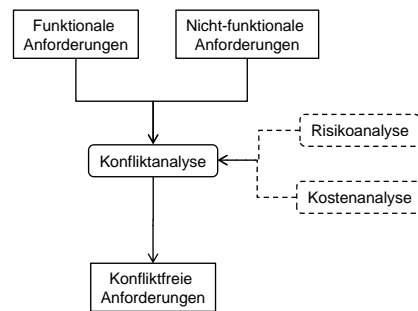
### Problem

Nicht-funktionale Anforderungen an ein System können u.U. nicht gleichzeitig erfüllt werden. Das klassische Beispiel sind Anforderungen an Integrität und schnelle Antwortzeit einer Systemfunktion. Beide sind nicht gleichzeitig bestmöglichst erfüllbar. Daher müssen Aktivitäten zur Aushandlung dieser Konflikte in ein Vorgehensmodell integriert werden.

### Lösung

Während der Anforderungsanalyse werden sowohl funktionale als auch nicht-funktionale Anforderungen erfasst sowie Systemgrenzen und Systemfunktionalität durch Nutzungsszenarien modelliert. In diesem Zusammenhang ist zusätzlich eine Konfliktanalyse der nicht-funktionalen Anforderungen durchzuführen. Dabei definiert das Prozessmuster nicht die konkrete Umsetzung dieser Aktivität. Diese ist abhängig von den Projektrisiken und Kostenschätzungen auszuwählen. Eine gute allgemeine Einführung in die Konfliktanalyse und das Konfliktmanagement gibt [Poh07]. Zur Konfliktanalyse nicht-funktionaler Anforderungen existieren verschiedene Ansätze (z.B. [CNYM00], [SL07] und [YT97] sowie als Überblick [R.C05] und [Lap05]).

## Struktur



## Aktivitäten

Das Muster beinhaltet nur eine Aktivität:

*Konfliktanalyse:* Die Konfliktanalyse dient dem Aufdecken und Lösen von widersprüchlichen Anforderungen an ein System. Dazu werden die nicht-funktionalen und funktionalen Anforderungen im Kontext der Systemmodelle analysiert. Die Analyse kann, wie bereits diskutiert, mit unterschiedlichen Techniken umgesetzt werden.

## Produktartefakte

*Input:* nicht-funktionale Anforderungen, funktionale Anforderungen, Systemmodelle

*zusätzlicher Input:* Ergebnisse einer Risikoanalyse, Ergebnisse einer Kostenanalyse

*Output:* konfliktfreie nicht-funktionale Anforderungen

## Kontext

Die Analyse möglicher Konflikte der nicht-funktionalen Anforderungen ist in das Konfliktmanagement eingebunden. Der Hauptakteur ist dabei der Anforderungsanalyst. Bestehende Konflikte sind durch geeignete Strategien aufzulösen. [Poh07] unterscheidet folgende Strategien: Verhandlung, Kreative Lösung und Entscheidung, wobei Verhandlung und Entscheidung für die Auflösung von Konflikten nicht-funktionaler Anforderungen empfohlen wird. Erstere ist durch die Optimierbarkeit der NFE ermöglicht. Da es aber keine vollständige Konfliktfreiheit aller Stakeholder-Anforderungen geben kann, ist auch die Strategie der Entscheidung einzusetzen. Die Konfliktlösung selbst ist nachvollziehbar zu dokumentieren.

## In Beziehung stehende Muster

*Explizite Spezifikation der NFE:* Ohne die explizite Spezifikation der nicht-funktionalen Systemanforderungen ist eine Analyse möglicher Konflikte zwischen verschiedenen Anforderungen nicht möglich.

## A.4 Verifikationsschleife

### Name und Klassifikation

Verifikationsschleife (Verification Loop)

Aktivitätsmuster, Prozessmuster der Anwendungsentwicklung

### Kurzbeschreibung

Die Verifikation von Modellen durchläuft immer die gleiche Reihenfolge von abstrakten Aktivitäten. Diese Reihenfolge ist unabhängig von den konkret eingesetzten Verifikationstechniken. Das Muster beschreibt die einzelnen Aktivitäten, deren Artefakte und Zusammenhänge aus einer allgemeingültigen Sicht.

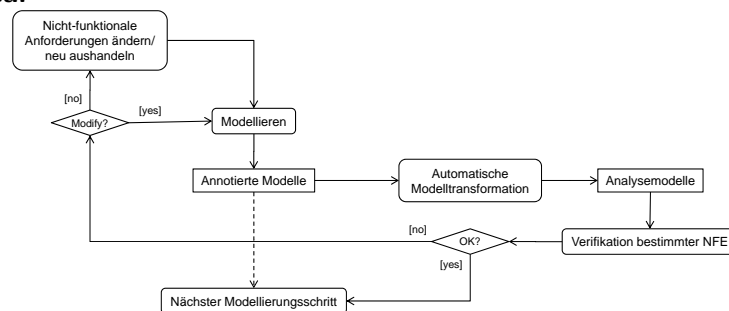
### Problem

Das zu lösende Problem ist die Integration von Verifikationstechniken in den Entwicklungszyklus, wobei Verifikationstechniken meist andere Sprachen als die Entwicklungsmodelle benötigen. Der Einsatz der Verifikation sollte im Bereich der Entwicklung kritischer Systeme ohne großen Lern- und Zeitaufwand durch die Entwickler möglich sein.

### Lösung

Die Lösung ist eine automatisierte Übersetzung des Entwicklungsmodells in das Analysemodell und die Rücktransformation der Analyseergebnisse in die Entwicklersicht. Dadurch erfolgt die Verifikation der nicht-funktionalen Eigenschaften für den Entwickler transparent. Dabei kann das Muster nicht nur für Modelle im klassischen Sinn eingesetzt werden. Die gleiche Aktivitätenfolge gilt auch für die Verifikation von Code. Da Code ebenfalls eine modellhafte Beschreibung der Wirklichkeit ist, fällt dieses Anwendungsszenario ebenfalls unter dieses Muster.

### Struktur



### Aktivitäten

*Automatische Modelltransformation:* Die Entwicklersicht des Modells ist in ein verifizierbares Modell zu übersetzen. Dabei ist auf die korrekte Abbildung der lokalen nicht-funktionalen Eigenschaften des Systems zu achten.

*Verifikation des Modells:* Das Modell ist auf die Erfüllung der spezifizierten globalen nicht-funktionalen Anforderungen zu verifizieren.



*Überprüfung des Verifikationsergebnisses:* Bei erfolgreicher Verifikation kann in der Entwicklung fortgeschritten werden. Bei nicht erfolgreicher Verifikation muss folgende Aktivität ausgeführt werden:

*Überprüfung, ob Modell angepasst werden kann:* Bei Fehlern im Modell muss geprüft werden, ob das Modell anpassbar ist, oder ob sogar die Anforderungen neu ausgehandelt werden müssen.

*Änderung der Anforderungen im Pflichtenheft:* Schlägt die Verifikation fehl und kann der Konflikt innerhalb der Modellierung nicht aufgelöst werden, sind die Anforderungen mit hoher Wahrscheinlichkeit nicht erfüllbar und daher mit dem Kunden neu auszuhandeln.

### Produktartefakte

*Input:* Modell/ Code mit nicht-funktionalen Eigenschaften

*Output:* Entscheidung über Notwendigkeit des Reengineerings des Models/ Codes bzw. neue Aushandlung der globalen nicht-funktionalen Anforderungen

### Kontext

Die Verifikation eines Modells wird entweder durch ein Werkzeug oder manuell durch eine Person durchgeführt. Es werden also zusätzliche Ressourcen zur Ausführung benötigt. Im Idealfall kann die Verifikation voll automatisiert und damit für den Entwickler transparent durchgeführt werden. In der Realität werden zumindest die Entscheidungen über eventuell notwendige Änderungen des Modells bzw. der Anforderungen durch reale Personen getroffen.

### In Beziehung stehende Muster

*NFE-spezifische Sprache:* Die explizite Beschreibung der nicht-funktionalen Anforderungen ist eine unabdingbare Voraussetzung für deren Verifikation.

## A.5 Additive Verifikation der NFE

### Name und Klassifikation

Additive Verifikation der NFE (Additive NFR-Verification)

Aktivitätsmuster, Prozessmuster der Anwendungsentwicklung

### Kurzbeschreibung

Das Muster der additiven Verifikation der NFE beschreibt die Möglichkeit, modulare Verifikationen nicht-funktionaler Eigenschaften auf unterschiedlichen Abstraktionsebenen zu einer Verifikation des Gesamtsystems zusammenzufassen.

### Problem

Bei der Entwicklung lauffzeitkritischer Systeme ist es unbedingt erforderlich, die Erfüllung der geforderten nicht-funktionalen Eigenschaften zu überprüfen. Innerhalb des Entwick-

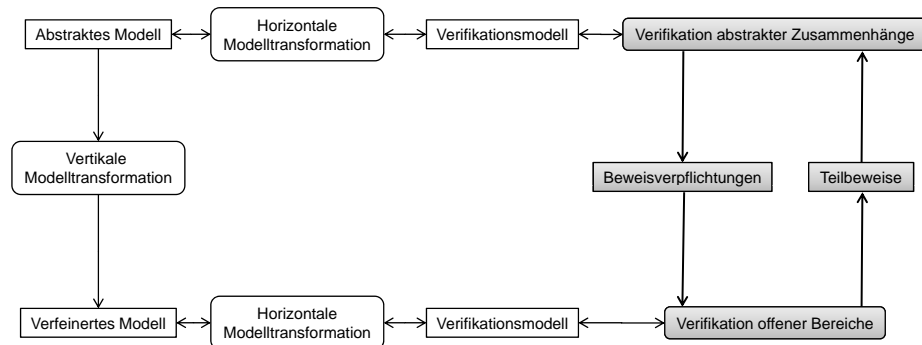
lungszyklus ist diese Überprüfung so früh wie möglich vorzusehen. Konkrete Laufzeitwerte stehen allerdings in den frühen Phasen nicht zur Verfügung.

### Lösung

Um Modelle schon in den frühen Phasen der Entwicklung auf Modellierungsfehler hinsichtlich der NFE zu überprüfen, werden Annahmen über konkrete Laufzeiteigenschaften eingesetzt. Diese Annahmen sind in den späteren Entwicklungsphasen zu überprüfen. Es entstehen also Beweisverpflichtungen, welche durch Teilbeweise des Systems überprüft werden müssen. In Kapitel 4.3 wurde dies am Beispiel von Annahmen für die Ausführungszeiten von Methoden auf Modellebene und deren Überprüfung durch Worst Case Execution Time (WCET) - Analysen der ausführbaren Software erläutert. Diese Kombination von Analysen ist übertragbar auf die Verknüpfung von Verifikationen verschiedener Aspekte eines Systems.

### Struktur

Das Muster selbst ist im grau hinterlegten Bereich dargestellt. Die gesamte Abbildung zeigt dessen Einordnung in den Ablauf der Systemmodellierung.



### Aktivitäten

*Verifikation:* beinhaltet die Verifikation verschiedener nicht-funktionaler Eigenschaften eines Systemmodells. Die Aktivität ist dieselbe wie im Muster „Verifikationsschleife“.

*Dokumentation und Weitergabe der Beweise:* Sowohl die Beweisverpflichtungen als auch bereite geführte Teilbeweise sind in geeigneter Form zu erfassen, zu dokumentieren und an entsprechende Bearbeiter weiterzugeben. Erst dadurch kann aus der Verifikation einzelner Systemeigenschaften die Verifikation des Gesamtsystems gefolgert werden.

### Produktartefakte

*Input:* Verifikationsmodelle verschiedener Abstraktionsebenen mit Angabe expliziter Maße von nicht-funktionalen Eigenschaften

*Output:* Beweisverpflichtungen, Teilbeweise

### Kontext

Der Einsatz dieses Musters erfordert viel Fachwissen bezüglich geeigneter Verifikationstechniken und deren Kombinierbarkeit. Daher ist die Expertenrolle „Verifikationsexperte“ mit der Planung und Umsetzung zu beauftragen. Zu den Aufgaben zählt sowohl die Planung und Dokumentation der Verifikationskette und der Kommunikation der Artefakte als auch die technische Umsetzung in verschiedenen Verifikationsschleifen. Bei der Planung muss sich der Verifikationsexperte mit den beteiligten Rollen abstimmen. Außerdem sind die Kosten der Verifikation gegen die zu analysierenden Risiken abzuwägen.

### In Beziehung stehende Muster

*Verifikationsschleife:* Ohne die Integration mehrerer Verifikationstechniken ist die additive Verifikation nicht möglich.

*NFE-spezifische Sprache:* Die explizite Beschreibung der nicht-funktionalen Anforderungen ist eine unabdingbare Voraussetzung für deren Verifikation.

*Transformation der NFE-Spezifikation:* Um die verschiedenen Beweise verbinden zu können ist die Definition der Abbildungsvorschriften zwischen den verschiedenen Modellen notwendig.

## A.6 Transformation der NFE-Spezifikationen

### Name und Klassifikation

Transformation der NFE-Spezifikationen (Transformation of NFR-Specifications)  
Aktivitätsmuster, Prozessmuster der Anwendungsentwicklung

### Kurzbeschreibung

Das Muster beschreibt die Notwendigkeit der Transformation der NFE in andere Modelle sowie die dafür notwendigen Artefakte und Aktivitäten.

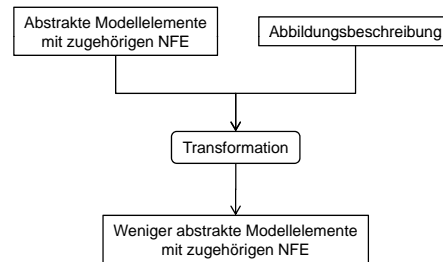
### Problem

Um die systematische Behandlung der spezifizierten nicht-funktionalen Eigenschaften zu gewährleisten, ist es notwendig diese innerhalb der Modellierung des Systems sowohl auf den Abstraktionsgrad verfeinerter Systemmodelle anzupassen, als auch die Eigenschaften in speziell für Analysen genutzte Modellsprachen zu übersetzen.

### Lösung

Dem Entwickler werden die notwendigen Abbildungsvorschriften der NFE zur Verfügung gestellt. Dabei werden Vorschriften zur horizontalen Abbildung der NFE zwischen verschiedenen Modellierungssprachen (z.B. Systemmodellierung und ausführbare Automaten zur Verifikation oder Testmodelle) und Vorschriften zum Schließen einer Abstraktionslücke bei der vertikalen Verfeinerung des Systems während der fortschreitenden Systementwicklung unterschieden. Es wird die Aktivität der *Transformation* in das Vorgehensmodell integriert, welche die Abbildung der expliziten Maße und damit die Anwendung der Abbildungsvorschriften fordert. Der Einsatz der Abbildungsvorschriften ist durch geeignete Werkzeuge zu unterstützen.

## Struktur



## Aktivitäten

Dieses Muster beinhaltet nur eine Aktivität:

*Transformation:* der NFE unter Nutzung der Abbildungsvorschriften, welche in der Abbildungsbeschreibung spezifiziert sind. Bei der Abbildung nicht-funktionaler Eigenschaften können Fälle auftreten, bei denen eine 1:n-Zuordnung möglich ist (siehe NFE-Prozessmuster „NFE-spezifische Abbildungsbeschreibung“). Ist dies der Fall, liegt die Auswahl der anzuwendenden Vorschrift im Entscheidungsspielraum des Anwenders.

## Produktartefakte

*Input:* NFE im Ausgangsmodell, Abbildungsbeschreibung

*Output:* NFE im Zielmodell

Entsprechend der beiden Haupteinsatzbereiche des Musters können Ausgangsmodell (AM) und Zielmodell (ZM) wie folgt ausgeprägt sein:

- Horizontale Transformation
  - AM: Systemmodell
  - ZM: Analyse-(Verifikations-)modell
- Vertikale Transformation
  - AM: abstraktes Systemmodell
  - ZM: weniger abstraktes Systemmodell

## Kontext

Dieses Muster findet in verschiedenen Kontexten Anwendung. Wie bereits beschrieben, wird zwischen vertikaler und horizontaler Transformation unterschieden. Demnach werden Transformationen von verschiedenen Rollen angestoßen. Dies sollte direkt aus der Entwicklungsumgebung möglich sein.

Ein NFE-spezifisches Problem bei der Transformation ist die angesprochene Möglichkeit der 1:n-Abbildung von Eigenschaften. Diese sind dem Entwickler nachvollziehbar zu erklären und zur Auswahl anzubieten. Die Beschreibung des Musters „NFE-spezifische Abbildungsbeschreibung“ enthält dazu ein Beispiel.

### In Beziehung stehende Muster

*NFE-spezifische Abbildungsbeschreibung:* Ohne die Bereitstellung NFE-spezifischer Abbildungsbeschreibungen ist das Muster nicht anwendbar.

## A.7 Analyse der Operationalisierung

### Name und Klassifikation

Analyse der Operationalisierung (Analysis of NFR-Operationalization)

Aktivitätsmuster, Prozessmuster der Anwendungsentwicklung

### Kurzbeschreibung

Das Muster beschreibt die Aktivitäten, die zur Konkretisierung von unterspezifizierten Anforderungen (Kapitel 2.1) notwendig sind.

### Problem

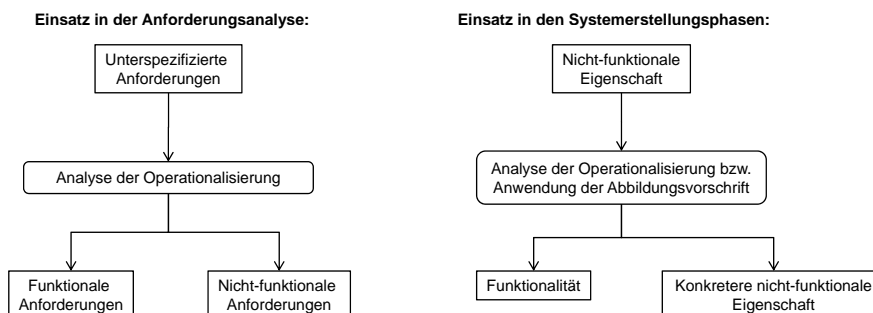
Einige nicht-funktionale Eigenschaften werden innerhalb der Anforderungsanalyse meist nur implizit oder informell erfasst. Dazu gehören Nutzaussagen wie: „Das System muss sicher gegenüber Angriffen von außen sein.“ Derartige Anforderungen sind keine rein nicht-funktionalen Anforderungen, sondern werden zum Teil durch Funktionalität umgesetzt.

### Lösung

Solche Eigenschaften werden als unterspezifizierte Anforderungen bezeichnet. Sie sind durch funktionale bzw. nicht-funktionale Eigenschaften zu konkretisieren. Dies erfolgt durch eine entsprechende Analyse der Operationalisierbarkeit der Eigenschaften.

Es können zwei Einsatzszenarien unterschieden werden. Während der Anforderungsanalyse erfolgt die Analyse der Operationalisierbarkeit durch den Menschen (Anforderungsanalyst). Im Gegensatz dazu ist die Operationalisierbarkeit der NFE einer Domäne in den vertikalen Abbildungsvorschriften zu integrieren. In diesem Fall wird die Operationalisierung während der Modelltransformation bzw. der Codegenerierung durch die entsprechenden Werkzeuge realisiert.

### Struktur



### Aktivitäten

*Analyse der Operationalisierung:* Die Analyse der Operationalisierung dient der Abgrenzung gegen die quantitativen Maße einer Funktionalität. Diese Aktivität kann durch verschiedene Techniken umgesetzt werden. In Kapitel 4.1 wurden einige Ansätze vorgestellt.

*Anwendung der Abbildungsvorschrift (alternative Aktivität):* Wurde die Operationalisierung der im Systemmodell spezifizierten NFE vorab in der NFE-spezifischen Abbildungsvorschrift definiert, ist diese während der Modelltransformation einzusetzen.

### Produktartefakte

*Input:* Dokumente der Anforderungsanalyse bzw. auf tieferer Ebene Systemmodelle mit spezifizierten NFE

*Output:* dem Abstraktionsgrad entsprechend konkretisierte Spezifikation funktionaler bzw. nicht-funktionaler Eigenschaften

### Kontext

Die Anwendung dieses NFE-Prozessmusters trägt wesentlich zur systematischen Präzisierung der NFE durch deren Abbildung auf funktionale Eigenschaften bzw. systemnahe nicht-funktionale Eigenschaften bei. Um solche Unterscheidungen zu treffen, gibt es Ansätze, wie z.B. das NFR-Framework [CNYM00]. Dieses unterstützt Entwurfsentscheidungen zur Umsetzung nicht-funktionaler Anforderungen in das tatsächliche System. Es basiert auf der Notation von „softgoals“ zur Repräsentation nicht-funktionaler Anforderungen. Diese können unpräzise und subjektiv sein. Softgoals stehen sowohl untereinander in Beziehung als auch zu „operationalisations“. Solche „Operationalisierungen“ spezifizieren mögliche Realisierungen für softgoals.

Wie bereits in Lösung und Struktur dargestellt, ist der Einsatz des Musters in zwei verschiedenen Projektphasen zu unterscheiden.

### In Beziehung stehende Muster

*Explizite Spezifikation der NFE:* Die explizite Spezifikation wird durch die Anwendung dieses Musters unterstützt.

*NFE-spezifische Abbildungsbeschreibung:* Zur Vermeidung von Inkonsistenzen und Fehlern bei Modellierungsentscheidungen, sind die möglichen Operationalisierungen der nicht-funktionalen Eigenschaften während der Domänenanalyse zu klären und in der NFE-spezifischen Abbildungsbeschreibung zu spezifizieren.

## A.8 Dekomposition der NFE

### Name und Klassifikation

Dekomposition nicht-funktionaler Eigenschaften (Decomposition of non-functional Requirements)

Aktivitätsmuster, Prozessmuster der Anwendungsentwicklung

### Kurzbeschreibung

Globale NFE-Anforderungen sind während der Entwicklung explizit auf mehrere Subsysteme bzw. Komponenten aufzuteilen. Dadurch werden die globalen Anforderungen auf lokale Eigenschaften einzelner Systemteile abgebildet.

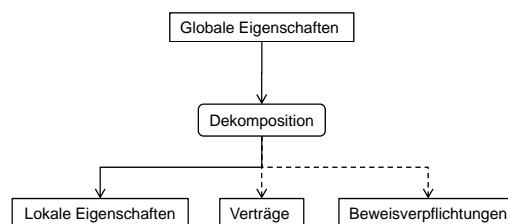
### Problem

Nicht-funktionale Eigenschaften werden in der Anforderungsanalyse als Beschränkungen der Dienste bzw. Funktionalität eines Systems definiert. Diese müssen bei der funktionalen Zerlegung des Systems adäquat dekomponiert werden. Die Problematik der NFE-Dekomposition liegt dabei in den konkreten Maßen, die für die expliziten Beschreibungen notwendig sind, da zum Zeitpunkt der Zerlegung die konkreten Bedingungen zur Laufzeit (z.B. Worst Case Execution Times von Methoden) noch nicht bekannt sind.

### Lösung

Globale Anforderungen sind durch entsprechende Methoden innerhalb der Anforderungsanalyse in lokale Anforderungen zu überführen. Bei deren Zerlegung in lokale Eigenschaften von Komponenten und Methoden sind Annahmen zu treffen, die im weiteren Verlauf der Entwicklung zu beweisen bzw. zu überprüfen sind. Über die Einhaltung der globalen Anforderungen durch die lokalen Systemeigenschaften sind Verträge zu spezifizieren.

### Struktur



### Aktivitäten

*Dekomposition:* Die Dekomposition beinhaltet die Identifikation der globalen Eigenschaften und deren Zerlegung in lokale Eigenschaften. Damit verbunden ist die explizite Zuordnung zu einzelnen Schnittstellen.

### Produktartefakte

*Input:* funktionale Systemmodelle und globale NFE

*Output:* lokale NFE

*zusätzlicher Output:* Sind die lokalen NFE als Annahmen spezifiziert entstehen zusätzlich Beweisverpflichtungen. Ist das spätere Beweisen der Annahmen nicht möglich, sind Verträge über den Zusammenhang der geforderten NFE zu spezifizieren und deren Erfüllung bei der späteren Komposition zu überprüfen (z.B. durch Testen, Simulieren, Reviews).

### Kontext

Dieses Muster beschreibt die gegenläufige Aktivität zur Komposition der NFE. Daher sind diese beiden Muster als Einheit zu betrachten und deren Kontext wird für beide im Muster Komposition der NFE diskutiert.

Wurden während der Dekomposition Beweisverpflichtungen festgelegt, gehen diese als Input in das Muster „Additive Verifikation der NFE“ ein.

Die Spezifikation der Verträge ist in den Modellierungswerkzeuge entsprechend zu unterstützen.

### In Beziehung stehende Muster

*Komposition nicht-funktionaler Eigenschaften:* Durch die Komposition der vom System erbrachten Maße der lokalen Eigenschaften wird die Erfüllung der globalen Eigenschaften des Systems überwacht. Die Komposition ist damit eine gegenläufige Aktivität zur Dekomposition.

*Explizite Spezifikation nicht-funktionaler Eigenschaften:* Die explizite Beschreibung der nicht-funktionalen Anforderungen ist eine unabdingbare Voraussetzung für deren Dekomposition.

*Analyse der Operationalisierung:* Durch die Dekomposition der NFE können Erkenntnisse über die Operationalisierbarkeit spezifischer nicht-funktionaler Anforderungen gewonnen werden.

## A.9 Komposition der NFE

### Name und Klassifikation

Komposition nicht-funktionaler Eigenschaften (Composition of non-functional requirements)

Aktivitätsmuster, Prozessmuster der Anwendungsentwicklung

### Kurzbeschreibung

Zur Überwachung der globalen Anforderungen müssen die Annahmen, die bei der Dekomposition für die lokalen Eigenschaften festgelegt wurden, überprüft werden.

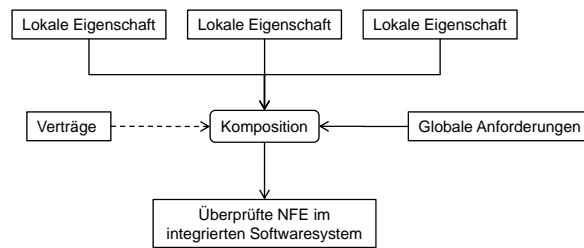
### Problem

Die lokalen nicht-funktionalen Eigenschaften können während der Dekomposition nur als Annahmen bzw. in Form von Verträgen spezifiziert werden. Diese müssen im Kontext der globalen Anforderungen überprüft werden.

### Lösung

Es wird eine Aktivität *Komposition* der NFE in den Entwicklungsprozess integriert. Durch die Komposition, also Integration, von Systemteilen werden aus den lokalen NFE globale. Die Aktivität fordert nun, dass während der Integration der Subsysteme bzw. Komponenten in das Gesamtsystem zu überprüfen ist, ob die tatsächlichen Werte der lokalen nicht-funktionalen Eigenschaften im Gesamtsystem die globalen nicht-funktionalen Anforderungen erfüllen.



**Struktur****Aktivitäten**

*Komposition:* Innerhalb der Komposition wird überprüft, ob die lokalen NFE von Komponenten eines (Teil-)Systems im Zusammenspiel die globalen nicht-funktionalen Anforderungen an dieses System erfüllen.

**Produktartefakte**

*Input:* lokale NFE, globale nicht-funktionale Anforderungen

*zusätzlicher Input:* Verträge über NFE

*Output:* auf die Erfüllung der nicht-funktionalen Anforderungen überprüfetes Softwaresystem bzw. Teilsysteme davon

**Kontext**

Die Komposition der NFE mit dem Ziel der Überprüfung der Erfüllung der globalen nicht-funktionalen Anforderungen durch die lokalen NFE ist Bestandteil des Integrationstests. Beteiligt sind daher sowohl Entwickler als auch Prüfexperten. Das NFE-Prozessmuster stellt die gegenläufige Aktivität zu der im Muster „Dekomposition“ beschrieben dar. Während in der Dekomposition Verträge über die dekomponierten Eigenschaften spezifiziert werden, werden diese Verträge nun auf deren Erfüllung überprüft.

Innerhalb der Komposition sind sowohl verschiedene Vertragsarten als auch verschiedene Ebenen der NFE und damit deren Verträge zu unterscheiden. Ebenen auf denen Verträge über NFE möglich sind, sind Nutzer-, Anwendungs- und Laufzeiteigenschaften. Dementsprechend existieren unter anderem Verträge zwischen Nutzer und Anwendung sowie zwischen Anwendung und Plattform. Weitere Vertragsarten über NFE sind in [ZR04] beschrieben.

**In Beziehung stehende Muster**

*Dekomposition der NFE:* Die Komposition überwacht die Erfüllung der lokalen NFE, welche während der Dekomposition bestimmt wurden.

*Inkrementelle Entwicklung der kritischen Systembereiche:* Die Komposition ist Teil des Integrationstests, welcher in diesem Muster gefordert wird.

## A.10 Entwicklung entlang des kritischen Pfades

### Name und Klassifikation

Entwicklung entlang des kritischen Pfades (Critical Path Development)

Phasenmuster, Prozessmuster der Anwendungsentwicklung

### Kurzbeschreibung

Das Prozessmuster beschreibt das Vorgehen einer systematischen Entwicklung beginnend mit den laufzeitkritischen Szenarien.

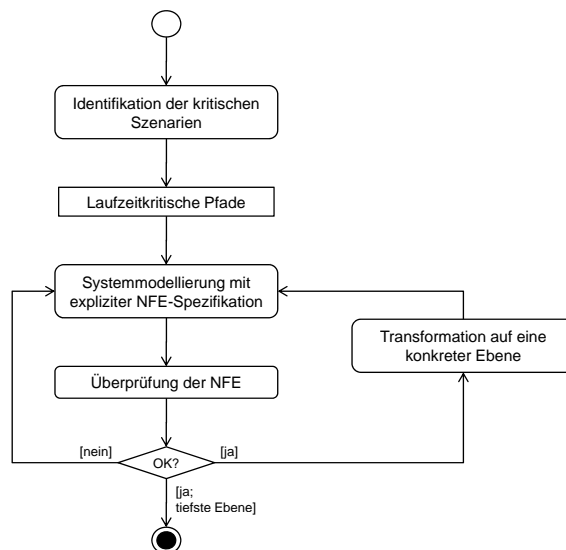
### Problem

Laufzeitkritische nicht-funktionale Eigenschaften sind derartig mit der Systemfunktionalität verwoben, dass es unzureichend ist, diese im Nachhinein zu integrieren. Vielmehr sind diese Eigenschaften während der Systementwicklung durchgängig und systematisch zu behandeln, um die Erfüllung zu gewährleisten.

### Lösung

Es wurden bereits mehrere NFE-Aktivitätsprozessmuster eingeführt, welche typische Aufgaben im Kontext der NFE-Realisierung beschreiben. Die Muster selbst sind in verschiedenen Phasen der Softwareentwicklung einsetzbar. Durch deren Komposition zu einem NFE-Phasenprozessmuster, ist es möglich eine Entwicklungsphase zur systematischen und durchgängigen Behandlung der NFE zu beschreiben.

### Struktur



### Aktivitäten

Diese Muster ist eine Komposition folgender NFE-Prozessmuster für Aktivitäten, welche eine Phase der Entwicklung definieren:

- 1) *Identifikation der kritischen Szenarien*

- 2) *Explizite Spezifikation der NFE*
- 3) *Verifikationsschleife*
- 4) *Transformation der NFE-Spezifikation*

### **Produktartefakte**

Da dieses Phasenprozessmuster aus den vorab genannten Aktivitätsprozessmustern zusammengesetzt ist, arbeitet es mit den darin beschriebenen Artefakten.

### **Kontext**

Es gelten die Bedingungen, welche in den einzelnen Mustern diskutiert werden. Bedingt durch die Optimierbarkeit der NFE ist zusätzlich ein Erfüllungskriterium als Abbruchbedingung für eine Entwicklungsphase in Form eines konkreten Wertes für die jeweilige nicht-funktionale Eigenschaft festzulegen.

### **In Beziehung stehende Muster**

Zusätzlich zu den bereits im Phasenmuster enthaltenen, können alle anderen definierten NFE-Aktivitätsmuster eingesetzt werden.

## **A.11 Inkrementelle Entwicklung der kritischen Systembereiche**

### **Name und Klassifikation**

Inkrementelle Entwicklung der kritischen Systembereiche (Incremental Development of Critical Components)

Phasenmuster, Prozessmuster der Anwendungsentwicklung

### **Kurzbeschreibung**

Für die Entwicklung laufzeitkritischer Systeme wird die Anwendung eines inkrementellen Vorgehensmodells vorgeschlagen. Dadurch ist es möglich, den Einfluss neuer Funktionalität auf die Erfüllung der geforderten nicht-funktionalen Laufzeiteigenschaften systematisch zu überwachen und bei Überschreitung der festgelegten Maße entgegen steuern zu können.

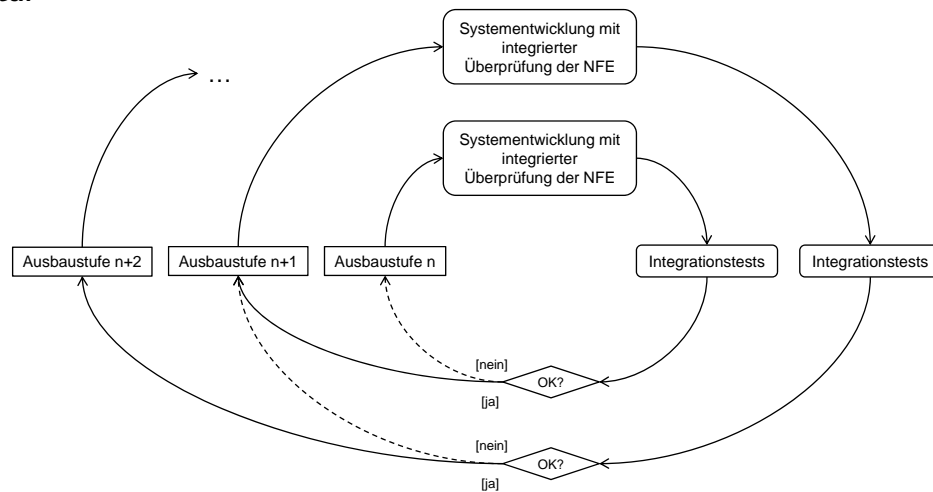
### **Problem**

Selbst wenn die Entwicklung mit einem kritischen Szenario begonnen wird, ist die Erfüllung nicht-funktionaler Laufzeiteigenschaften eines Systems durch das Hinzufügen neuer Funktionalität nicht zwangsläufig sichergestellt. So benötigt beispielsweise jede zusätzliche Funktion Rechenzeit, die sich wiederum auf zeitbasierte Eigenschaften anderer Systemfunktionen auswirken können.

### **Lösung**

Als Lösung wird ein inkrementelles Vorgehen zur Entwicklung der laufzeitkritischen Komponenten bzw. Subsysteme vorgeschlagen. Außerdem werden zwischen den einzelnen Inkrementen Integrationstests mit dem Gesamtsystem gefordert.

## Struktur



## Phasen

Innerhalb des Musters können zwei Phasen unterschieden werden:

*Systementwicklung mit integrierter Überprüfung der NFE:* Innerhalb der einzelnen Ausbaustufen der laufzeitkritischen Software wird jeweils das Phasenmuster „Entwicklung entlang des kritischen Pfades“ durchlaufen. Dabei werden die nicht-funktionalen Eigenschaften des Systems lokal innerhalb des gerade entwickelten Subsystems bzw. der Komponente überprüft.

*Integrationstests:* Durch den Integrationstest vor der Erweiterung der Systemfunktionalität in der nächsthöheren Ausbaustufe wird sichergestellt, dass die verschiedenen voneinander abhängigen Komponenten des Gesamtsystems die geforderten globalen nicht-funktionalen Anforderungen im Zusammenspiel erfüllen.

## Produktartefakte

*Input:* Artefakte einer Iterationsstufe des Systems

*Output:* Artefakte der nächst folgenden Iterationsstufe des Systems

## Kontext

Dieses Phasenmuster dient der Entwicklung solcher Systembereiche, die bezüglich ihrer nicht-funktionalen Anforderungen als kritisch identifiziert wurden. Durch den inkrementellen Ausbau dieser Systeme sowie die geforderten Integrationstests mit den unkritischen Systemteilen, können negative Beeinflussungen der geforderten NFE frühzeitig erkannt werden und entsprechende Gegenmaßnahmen ergriffen werden.

## In Beziehung stehende Muster

Innerhalb dieses Phasenmusters können alle definierten Aktivitätsmuster eingesetzt werden.

## A.12 NFE-spezifische Sprache

### Name und Klassifikation

NFE-spezifische Sprache (NFR-specific Language)

Aktivitätsmuster, Prozessmuster der Domänenentwicklung

### Kurzbeschreibung

Das Muster beschreibt die Aktivitäten, die zur Bereitstellung NFE-spezifischer Sprachen notwendig sind.

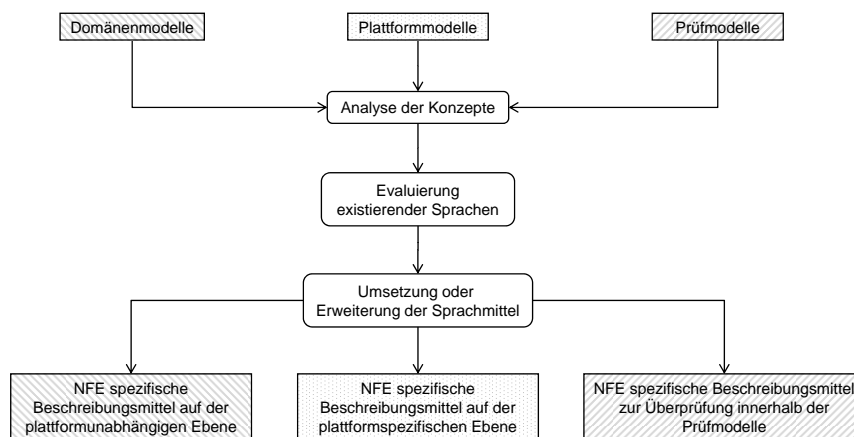
### Problem

Um laufzeitkritische nicht-funktionale Eigenschaften durchgängig zu behandeln, ist es unerlässlich durchgängige Beschreibungstechniken bereitzustellen. Diese müssen dem Entwickler die Spezifikation nicht-funktionaler Laufzeiteigenschaften in geeigneter intuitiver Form ermöglichen sowie die Angabe konkreter Maße für diese Eigenschaften unterstützen.

### Lösung

Die Sprachen sind je nach Einsatzzweck aufbauend auf der Analyse der Konzepte der Domänen-, Plattform- bzw. Prüfmodelle zu erstellen. Die Sprachen können entweder durch Erweiterung existierender Modellierungssprachen, z.B. durch die Definition von Profilen für UML-Modelle oder durch die Definition neuer Sprachen, z.B. durch die Definition von Metamodellen erfolgen. Außerdem ist zu überprüfen, ob es existierende Sprachen gibt, die den Anforderungen der speziellen nicht-funktionalen Eigenschaften genügen. Die Spezifikationsprache kann entweder generisch für alle NFE oder für eine spezifische Eigenschaft bzw. Gruppe von Eigenschaften definiert werden.

### Struktur



### Aktivitäten

*Analyse der Konzepte der Domänen-, Plattform- bzw. Prüfmodelle:* Voraussetzung für die Definition der Sprachkonzepte ist eine umfassende Analyse. Basis der Analyse

sind je nach Abstraktion der zu definierenden Sprache das Domänen-, Plattform- bzw. Prüfmodell. Der Schwerpunkt der Analyse liegt dabei auf dem Verständnis der Merkmale der nicht-funktionalen Eigenschaften.

*Evaluierung existierender Sprachen:* Vor der eigentlichen Sprachdefinition ist es eine vornehmliche Aufgabe des Spracharchitekten Sprachen Dritter zu suchen und zu evaluieren.

*Definition der Sprachkonzepte innerhalb einer Beschreibungssprache:* Auf Basis der geführten Analyse werden die notwendigen NFE-spezifischen Beschreibungssprachen für die Modellierung der nicht-funktionalen Eigenschaften innerhalb eines Softwareentwicklungsprozesses bereitgestellt.

### Produktartefakte

*Input:* Domänen-, Plattform- bzw. Prüfmodelle

*Output:* NFE-spezifische Beschreibsprachen für die plattformunabhängigen, plattform-spezifischen bzw. die Prüfmodelle

### Kontext

Für die Definition der notwendigen Sprachkonzepte sind der NFE-Wissensingenieur (Anhang B) bzw. dessen entsprechende Unterrollen zuständig. Da die erstellte Sprache ein wesentliches Mittel der Anwendungsentwicklung darstellt, ist eine enge Abstimmung mit allen Beteiligten zur Aufnahme der Anforderungen sowie zur Pflege der Sprache (Evaluierung, Verbesserung und Wartung) notwendig.

Ein Problem der Spezifikation nicht-funktionaler Eigenschaften ist die Komplexität der Maße. Im Kapitel 4.3 wurde als Beispiel die Spezifikation eines Maßes für Antwortzeit in der Sprache CQML<sup>+</sup> angegeben. Zu sehen war die Dreiteilung der Spezifikation in die Definition des Maßes an sich (**characteristic**), dessen Einschränkung auf einen konkreten Wert (**quality**) und die Bindung des Wertes an eine konkrete Komponente (**profile**). Während der erste Teil der Spezifikation mit den domänenspezifischen Sprachen in der modellgetriebenen Entwicklung vergleichbar und damit wiederverwendbar ist, sind die anderen anwendungsspezifisch. Dem Anwendungsentwickler sollte die Komplexität der Maßdefinition verborgen bleiben [RZ04b]. In [Ban04] wird vorgeschlagen, den Anwendungsentwicklern über eine Suchmaske an Hand textueller Beschreibungen von Eigenschaften die Auswahl einer geeigneten Maßdefinition zu erleichtern. Die komplexe Definition der Wertebereiche und der Semantik des Maßes bleibt vor ihm verborgen.

### In Beziehung stehende Muster

Die überwiegende Anzahl der *NFE-Prozessmuster der Anwendungsentwicklung* bauen auf der Existenz der NFE-spezifischen Sprachmittel auf, daher ist dieses Muster Voraussetzung für den Einsatz des durch die NFE-Prozessmuster erweiterten Vorgehensmodells.

## A.13 NFE-spezifische Abbildungsbeschreibung

### Name und Klassifikation

NFE-spezifische Abbildungsbeschreibung (NFR-specific mapping specification)

Aktivitätsmuster, Prozessmuster der Domänenentwicklung

### Kurzbeschreibung

Das Muster beschreibt die Aktivitäten, die zur Bereitstellung von NFE-spezifischen Abbildungsbeschreibungen notwendig sind.

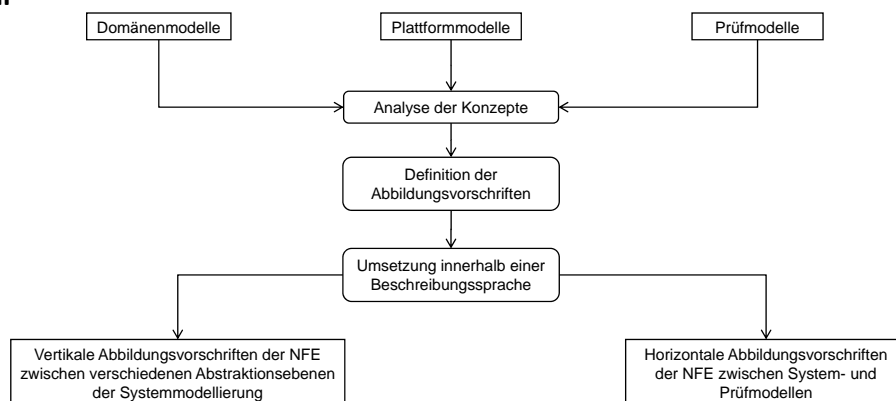
### Problem

Um laufzeitkritische nicht-funktionale Eigenschaften systematisch zu behandeln, ist es notwendig, durchgängige Beschreibungstechniken bereitzustellen.

### Lösung

Zur Überbrückung der Abstraktionslücken zwischen den einzelnen Darstellungen werden Abbildungsbeschreibungen spezifiziert und durch Transformations- bzw. Templatebeschreibungssprachen implementiert. Da die Abstraktionslücken sowohl zwischen den verschiedenen Ebenen der Systemmodellierung (vertikal) als auch zwischen System- und Prüfmodellen existieren, entsteht eine Menge von Abbildungsbeschreibungen.

### Struktur



### Aktivitäten

*Analyse der Konzepte der Domänen-, Plattform- bzw. Prüfmodelle:* Abhängig von der jeweils zu erstellenden Abbildungsbeschreibung müssen die zugehörigen Modelle auf deren zu übersetzende Konzepte untersucht werden.

*Definition der Abbildungsvorschriften:* Die analysierten Konzepte sind in Form von Abbildungsmechanismen zu spezifizieren. Diese beschreiben die Beziehungen zwischen zwei Konzepten der unterschiedlichen Abstraktionsebenen.

*Umsetzung innerhalb einer Beschreibungssprache:* Abhängig von der Art der Werkzeugunterstützung und der Art der Transformation (Modell-zu-Modell oder Modell-zu-

Code) sind entsprechende Beschreibungssprachen zu evaluieren und in diesen die definierten Abbildungsmechanismen zu implementieren.

### **Produktartefakte**

*Input:* Domänen-, Plattform- bzw. Prüfmodelle

*Output:* NFE-spezifische Abbildungsbeschreibungen zwischen den definierten Abstraktionsebenen der Systembeschreibungen während der Anwendungsentwicklung. Wie beschrieben, entsteht eine Menge aus horizontalen und vertikalen Abbildungsbeschreibungen.

### **Kontext**

Den Kern der Aufgaben innerhalb dieses Prozessmusters übernimmt der Transformationsarchitekt. Allein die Auswahl der Transformationssprache sollte in enger Zusammenarbeit mit dem NFE-Spracharchitekten erfolgen. Zur Erstellung der Abbildungskonzepte ist außerdem die Abstimmung mit den jeweiligen Rollen des NFE-Wissensingenieurs erforderlich.

### **In Beziehung stehende Muster**

*NFE-spezifische Sprache:* Die Entwicklung der Abbildungsbeschreibungen kann nur im engen Zusammenhang mit der Entwicklung der NFE-spezifische Sprachmittel erfolgen, da diese als Ein- bzw. Ausgabesyntax für die Implementierung der Abbildungen in einer konkreten Sprache dienen.

*NFE-Prozessmuster der Anwendungsentwicklung:* Die Abbildungsbeschreibungen und deren Umsetzung in Transformationssprachen ist Voraussetzung für den Einsatz der meisten der hier beschriebenen NFE-Prozessmuster der Anwendungsentwicklung. Nicht betroffen sind nur solche Muster, die vorwiegend während der Anforderungsanalyse eingesetzt werden.



## B Rollen und deren Aufgaben

Im Folgenden werden die Rollen der Anwendungs- und der Domänenentwicklung sowie deren Aufgaben bezüglich der durchgängigen Unterstützung nicht-funktionaler Systemeigenschaften beschrieben. Während der NFE-Domänenentwicklung erfolgt die Bereitstellung einer Spracharchitektur ähnlich der einer MDA-Architektur. Innerhalb der Anwendungsentwicklung werden nicht alle notwendigen Rollen aufgeführt, sondern nur solche, die im Zusammenhang zu dieser Arbeit stehen. Die Beschreibung der Rollen basiert teilweise auf [yAS04] und [GPR06].

### B.1 Anwendungsentwicklung

#### B.1.1 Software- und Businessanalyst

**Beschreibung:** Die Aufgabe der Analysten ist es, die funktionalen und nicht-funktionalen Anforderungen des Systems zu erfassen, zu analysieren und in der vorgeschriebenen Modellform aufzuschreiben. Damit dienen die Modelle, im Sinne dieser Arbeit vor allem die nicht-funktionalen Beschreibungen, nicht mehr nur der Kommunikation, sondern werden durch ihren formalen Charakter zur Transformation in weniger abstrakte Modelle verwendet.

**Ergebnisse (Work Products):** Analysemodelle des Systems mit integrierter expliziter Beschreibung der nicht-funktionalen Anforderungen

#### B.1.2 Softwarearchitekt

**Beschreibung:** Softwarearchitekten haben innerhalb des PROKRIS-Frameworks zusätzlich zu ihrer Arbeit neue Aufgaben zu erfüllen. So müssen sie die nicht-funktionalen Anforderungen an das System in globale und lokale Eigenschaften dekomponieren und diese explizit Schnittstellen innerhalb der Softwarearchitektur zuordnen. Ihre Aufgabe ist es außerdem, die Eigenschaften unter Nutzung der vorgegebenen Spezifikationstechniken explizit durch Maße zu präzisieren. Zur Unterstützung einer parallelen Entwicklung von Systemteilen müssen sie innerhalb der Architektur NFE-kritische von unkritischen Bereichen trennen. Die Architektur sollte so modelliert werden, dass unabhängige funktionale Schnittstellen existieren, die sich in ihren Qualitätsanforderungen nicht beeinflussen. An den Schnittstellen, an denen dies nicht möglich ist, sind Verträge über die NFE-Erfüllung zu spezifizieren. Eine weitere Aufgabe ist die Verifikation der spezifizierten nicht-funktionalen Eigenschaften der Modelle. Dazu steht ihm die

Ausführungsumgebung zur Verfügung, innerhalb der die Modellverifikation automatisiert abläuft.

**Ergebnisse (Work Products):** verifizierte Architekturmodelle mit integrierter Spezifikation der NFE, Verträge über NFE

### **B.1.3 Entwickler**

**Beschreibung:** Die Entwickler sind innerhalb des PROKRIS-Frameworks nicht nur für die Codegenerierung und Anpassung verantwortlich. Sie müssen außerdem den Code auf die Erfüllung der geforderten nicht-funktionalen Eigenschaften verifizieren. Dazu steht ihnen die Ausführungsumgebung des PROKRIS-Frameworks zur Verfügung.

**Ergebnisse (Work Products):** auf die Erfüllung der NFE verifizierter Code (annotiert mit Informationen zur Laufzeitumsetzung der NFE)

### **B.1.4 Prüfexperten für NFE**

**Beschreibung:** Aufgabe der Prüfexperten für NFE ist die Überprüfung der Erfüllung der nicht-funktionalen Anforderungen. Eine Unterrolle ist der Verifikationsexperte. Er ist für die Aktivitäten innerhalb der Verifikationsschleife sowohl auf Modell- als auch auf Codeebene verantwortlich. Im günstigsten Fall findet dies voll automatisiert und damit für den Entwickler transparent statt. Doch selbst dann sollte eine derartige Rolle zur Verfügung stehen, um bei fraglichen Analyseergebnissen beratend eingreifen zu können. Der Verifikationsexperte hat außerdem Aufgaben während der NFE-Domänenentwicklung zu erfüllen, die dort beschrieben sind.

**Ergebnisse (Work Products):** Ergebnisse der Überprüfung der nicht-funktionalen Eigenschaften eines Systemmodells

## **B.2 NFE-Domäneneentwicklung**

Die Hauptrolle der NFE-Domänenentwicklung ist der NFE-Wissensingenieur. Diese gliedert sich in folgende Unterrollen (Abbildung 4.10).

### **B.2.1 Domänenexperte**

**Beschreibung:** Der Domänenexperte ist Fachmann innerhalb der ausgewählten Anwendungsdomäne. In Zusammenhang mit dem PROKRIS-Framework handelt es sich dabei um kritische Systeme, beispielsweise eingebettete Systeme oder Software mit unmittelbarer Auswirkung auf die Sicherheit von Menschen. Er ist verantwortlich für die Abstraktion der in der Domäne vorliegenden Konzepte und bildet so die Ausgangsbasis für die Bereitstellung von Beschreibungstechniken (UML-Profile, Metamodelle, Transformationen)

durch die Architekten. Voraussetzung für die Erfüllung der Aufgaben ist gelebte Erfahrung in der Domäne und Abstraktionsvermögen zur Aufbereitung des Domänenwissens. Außerdem muss diese Rolle mit Modellierungstechniken und -werkzeugen sowie den verwendeten Modellierungssprachen und Transformationssprachen familiär sein.

**Ergebnisse (Work Products):** Analysemodelle der Domänenkonzepte

### B.2.2 Plattformexperte

**Beschreibung:** Der Plattformexperte ist verantwortlich für die Abstraktion von Fachwissen bezüglich der Realisierung der domänenspezifischen Anwendungen durch konkrete Technologien (Plattformen) zur Konservierung innerhalb des Frameworks. Diese Arbeit ist ebenfalls Voraussetzung für die Arbeit der Architekten.

**Ergebnisse (Work Products):** Analysemodelle der Plattformkonzepte

### B.2.3 Verifikationsexperte

**Beschreibung:** Der Verifikationsexperte ist verantwortlich für die Abstraktion von Wissen bezüglich der Verifikation verschiedenster nicht-funktionaler Laufzeiteigenschaften und deren Umsetzung in analysfähige Modelle.

**Ergebnisse (Work Products):** Mögliche Analysetechniken und zugehörige Modelle

### B.2.4 Spracharchitekt

**Beschreibung:** Auf Basis der Analysen der Expertenrollen und der daraus entstandenen Domänen-, Plattform- und Analysemodelle erstellt der Spracharchitekt die notwendigen Beschreibungssprachen für die anfallenden Modellierungsaufgaben. Dazu gehört z.B. die Erweiterung von UML-Diagrammen durch Profile oder die Erstellung von neuen Sprachen durch die Definition von Metamodellen zur Beschreibung nicht-funktionaler Eigenschaften. Außerdem ist es seine Aufgabe zu prüfen, ob existierende Spezifikationsmöglichkeiten (z.B. eine bestehende Maßbibliothek wie in [RZ07] beschrieben) verwendet werden kann. Eine weitere Aufgabe des Spracharchitekten ist die Auswahl einer geeigneten Sprache sowohl zur Überbrückung der Modellabstraktionsebenen als auch zur Beschreibung der Codegenerierungstemplates.

**Ergebnisse (Work Products):** Beschreibungsmittel für NFE-Domänenkonzepte einschließlich der nicht-funktionalen Eigenschaften, Transformationssprache, Templatebeschreibungssprache für Codegenerierung

**Abgeleitete Rollen** sind jeweils verantwortlich für:

- NFE-Maße
- Systemmodellierung
- Verifikation

### **B.2.5 Transformationsarchitekt**

**Beschreibung:** Aus den Ergebnissen der Analysen der verschiedenen Experten und der vom Spracharchitekten zur Verfügung gestellten Sprachen erstellt der Transformationsarchitekt die benötigten Transformationsbeschreibungen. Innerhalb des PROKRIS-Frameworks sind folgende Transformationen zu unterscheiden: vertikale Modell-zu-Modell Transformation (abstraktes Systemmodell zu weniger abstraktem, z.B. PIM zu PSM), horizontale Modell-zu-Modell Transformation (Systemmodell zu Analysemodell) und Modell-zu-Code Transformation. Neben der Bereitstellung der Beschreibungen gehört auch die Realisierung der entsprechenden Software, wie Transformatoren und Co-degeneratoren zu den Aufgaben des Transformationsarchitekten.

**Ergebnisse (Work Products):** Transformationsbeschreibungen und zugehörige Software

**Abgeleitete Rollen** sind jeweils verantwortlich für:

- horizontale Transformationen
- vertikale Transformationen

### **B.2.6 Methodenspezialist**

**Beschreibung:** Der Methodenspezialist ist für die Anpassung von existierenden Methoden innerhalb der PROKRIS-Prozessbibliothek verantwortlich. Er legt dazu die notwendigen Rollen, Aktivitäten und Artefakte fest. In der Anwendungsentwicklung steht er als Ansprechpartner zur Verfügung. Außerdem ist er dafür verantwortlich das Feedback der verschiedenen Stakeholder in die Weiterentwicklung und Verbesserung des Inhalts der PROKRIS-Prozessbibliothek einfließen zu lassen.

**Ergebnisse (Work Products):** Inhalt der PROKRIS-Prozessbibliothek

## C Ressourcenspezifikation

Das folgende Beispiel eines Ressourcenmodells ist eine etwas gekürzte Variante des Modells des SuReal-Demonstrators der Evaluierung. Die Kürzung erfolgte in den letzten Phasen der Entwicklung, das heißt die Ressourcen, welche zur Verifikation der Worst Case Execution Time im ausführbaren Code notwendig sind fehlen. Das Beispiel zeigt eine ausführlichere Gruppierung der Ressourcen (ab Zeile 3) als im Text der vorliegenden Arbeit. Für die Erklärung der verschiedenen Möglichkeiten der Datenflussmodellierung innerhalb der Beschreibung der einzelnen Werkzeuge wird auf [RSS09] verwiesen.

Listing C.1: Vereinfachtes Beispiel der Ressourcenspezifikation des Demonstrators

```
<?xml version="1.0" encoding="UTF-8"?>
2 <resourceDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:tool="http://www.example.org/Tool">
  <resourceGroup name="Analysis" ref="aa01 Notepad Wordpad AnalysisData" />
4 <category name="Development">
  <resourceGroup name="DeveloperTeam" ref="aa01 mm01 ">
6 <category name="PIM">
  <resourceGroup name="design" ref="Ameos_Data Ameos" />
8 <resourceGroup name="verification" ref="ff01 UPPAAL CruiseControlModel" />
  <resourceGroup name="transform" ref="LAST.OUTPUT Ameos2EclipseUML" />
10 </category>
  <category name="PSM">
12 <resourceGroup name="verification" ref="ff01 SymtaInputCruiseControl
    SymTAS.CommandLine" />
  <resourceGroup name="design" ref="Ameos_Data Ameos" />
14 <resourceGroup name="transform" ref="LAST.OUTPUT Ameos2Symtas" />
  <resourceGroup name="generation" ref="AntScript4CodeGeneration
    JamaicaCodeGenerator" />
16 </category>
  </resourceGroup>
18 </category>
  <resourceGroup name="PDM" ref="tt01">
20 <resourceGroup name="compile" ref="JamaicaTool JamaicaMainClass" />
  </resourceGroup>
22 </resourceGroup>
  <resourceList>
24 <resource id="Notepad" name="Notepad" type="program">
  <tool:Tool>
26 <tool:Name>Commandline-Tool Notepad</tool:Name>
  <tool:Description>The application Notepad.</tool:Description>
28 <tool:ParamresourceList xsi:type="tool:PLCommandline">
  <tool:ExecutionPath>C:\Windows\</tool:ExecutionPath>
30 <tool:ExecutionFile>notepad.exe</tool:ExecutionFile>
  <tool:Options>[INPUT]</tool:Options>
32 </tool:ParamresourceList>
  <tool:OutputURL>[CURRENTINPUT_URL]</tool:OutputURL>
34 <tool:AutomaticTool>false</tool:AutomaticTool>
  </tool:Tool>
36 </resource>
  <resource id="Wordpad" name="Wordpad" type="program">
```

```

38   <tool:Tool>
39     <tool:Name>Commandline-Tool Wordpad</tool:Name>
40     <tool:Description> The application Wordpad.</tool:Description>
41     <tool:ParamresourceList xsi:type="tool:PLCommandline">
42       <tool:ExecutionPath>C:\Programme\Windows NT\Zubehoer</tool:ExecutionPath>
43       <tool:ExecutionFile>wordpad.exe</tool:ExecutionFile>
44       <tool:Options>[INPUT]</tool:Options>
45     </tool:ParamresourceList>
46     <tool:OutputURL>[CURRENTINPUT_URL]</tool:OutputURL>
47     <tool:AutomaticTool>>false</tool:AutomaticTool>
48   </tool:Tool>
49 </resource>
50 <resource id="Ameos" name="Ameos" type="program">
51   <tool:Tool>
52     <tool:Name>Eclipse-Plugin-Tool Ameos</tool:Name>
53     <tool:Description>This is an Eclipse-Plugin starting Ameos.</tool:Description>
54     <tool:ParamresourceList xsi:type="tool:PLEclipsePlugin">
55       <tool:PluginID> org.surreal.eclipse.uppaaltools</tool:PluginID>
56       <tool:MethodClass> org.surreal.eclipse.uppaaltools.action.StartAmeosAction
57         </tool:MethodClass>
58       <tool:Method>runAmeos</tool:Method>
59       <tool:ParameterClass>java.net.URI</tool:ParameterClass>
60     </tool:ParamresourceList>
61     <tool:OutputURL>[CURRENTINPUT_URL]</tool:OutputURL>
62     <tool:AutomaticTool>>false</tool:AutomaticTool>
63   </tool:Tool>
64 </resource>
65 <resource id="Ameos2EclipseUML" name="Ameos2EclipseUML" type="program">
66   <tool:Tool>
67     <tool:Name> Automatic-Eclipse-Plugin Convert to UML</tool:Name>
68     <tool:Description> This tool converts a Ameos-System to an Eclipse
69       UML-file.</tool:Description>
70     <tool:ParamresourceList xsi:type="tool:PLEclipsePlugin">
71       <tool:PluginID> org.surreal.eclipse.uppaaltools</tool:PluginID>
72       <tool:MethodClass>
73         org.surreal.eclipse.uppaaltools.action.Ameos2EclipseUMLJobAction
74       </tool:MethodClass>
75       <tool:Method>runAutomatically</tool:Method>
76       <tool:ParameterClass>java.io.File</tool:ParameterClass>
77     </tool:ParamresourceList>
78     <tool:OutputURL>[CURRENTINPUT_X_EXT] \ [CURRENT_NAME].uml</tool:OutputURL>
79     <tool:AutomaticTool>true</tool:AutomaticTool>
80   </tool:Tool>
81 </resource>
82 <resource id="UPPAAL" name="UPPAAL" type="program">
83   <tool:Tool>
84     <tool:Name> Automatic-Eclipse-Plugin-Tool Verify with UPPAAL</tool:Name>
85     <tool:Description> This tool verifies a UML-model with
86       UPPAAL.</tool:Description>
87     <tool:ParamresourceList xsi:type="tool:PLEclipsePlugin">
88       <tool:PluginID> org.surreal.eclipse.uppaaltools</tool:PluginID>
89       <tool:MethodClass>
90         org.surreal.eclipse.uppaaltools.action.EclipseUML2UpaalJobAction
91       </tool:MethodClass>
92       <tool:Method>runAutomatically</tool:Method>
93       <tool:ParameterClass>java.io.File</tool:ParameterClass>
94     </tool:ParamresourceList>
95     <tool:OutputURL>[CURRENTINPUT_X_EXT].uppaal.xml</tool:OutputURL>
96     <tool:AutomaticTool>true</tool:AutomaticTool>
97   </tool:Tool>
98 </resource>
99 <resource id="Ameos2Syntas" name="Ameos2SymTA/S" type="program">

```

```

94  <tool:Tool>
    <tool:Name> Automatic-Eclipse-Plugin-Tool Convert to UML</tool:Name>
    <tool:Description> This tool converts a Ameos-System to an
        SymTA/S-file.</tool:Description>
96  <tool:ParamresourceList xsi:type="tool:PLEclipsePlugin">
    <tool:PluginID> org.sureal.eclipse.uppaaltools</tool:PluginID>
98  <tool:MethodClass>
        org.sureal.eclipse.uppaaltools.action.Ameos2EclipseUMLJobAction
        </tool:MethodClass>
    <tool:Method>runFibexGenerationAutomatically</tool:Method>
100  <tool:ParameterClass>java.io.File</tool:ParameterClass>
    </tool:ParamresourceList>
102  <tool:OutputURL>[CURRENTINPUT.X_EXT]\[CURRENTNAME].xml</tool:OutputURL>
    <tool:AutomaticTool>true</tool:AutomaticTool>
104 </tool:Tool>
</resource>
106 <resource id="SymTAS_CommandLine" name="SymTA/S Commandline" type="program">
    <tool:Tool>
108  <tool:Name>SymTAS</tool:Name>
    <tool:Description> This tools starts SymTA/S.</tool:Description>
110  <tool:ParamresourceList xsi:type="tool:PLCommandline">
    <tool:ExecutionPath>C:\Programme\Symtavision
        GmbH\SymTA-S</tool:ExecutionPath>
112  <tool:ExecutionFile>SymtaClient.exe</tool:ExecutionFile>
    </tool:ParamresourceList>
114  <tool:OutputURL>[CURRENTINPUT_URL]</tool:OutputURL>
    <tool:AutomaticTool>false</tool:AutomaticTool>
116 </tool:Tool>
</resource>
118 <resource id="JamaicaCodeGenerator" name="Jamaica-Codegenerator" type="program">
    <tool:Tool>
120  <tool:Name> AntTool for generation of Java-Code</tool:Name>
    <tool:Description> This tool generates Java-code of an
        UML-Model.</tool:Description>
122  <tool:ParamresourceList xsi:type="tool:PLAnt">
    <tool:Target>generate_source_code</tool:Target>
124  </tool:ParamresourceList>
    <tool:OutputURL>[CURRENTINPUT_URL]</tool:OutputURL>
126  <tool:AutomaticTool>false</tool:AutomaticTool>
    </tool:Tool>
128 </resource>
130 <resource id="JamaicaTool" name="JamaicaTool" type="program">
    <tool:Tool>
132  <tool:Name> Automatic JamaicaTool</tool:Name>
    <tool:Description> This tool compiles the generated code within the JamaicaVM
        to platform specific binary.</tool:Description>
    <tool:ParamresourceList xsi:type="tool:PLEclipsePlugin">
134  <tool:PluginID> com.aicas.jamaica.eclipse.sureal</tool:PluginID>
    <tool:MethodClass>
        com.aicas.jamaica.eclipse.sureal.JamaicaWorkflowStep</tool:MethodClass>
136  <tool:Method>buildApplication</tool:Method>
    </tool:ParamresourceList>
138  <tool:OutputURL>[WORKSPACE]\SuReal\[CURRENTNAME].exe</tool:OutputURL>
    <tool:AutomaticTool>true</tool:AutomaticTool>
140 </tool:Tool>
</resource>
142
144 <resource id="AnalysisData" name="Analysis.txt" type="data"
    url="[WORKSPACE]\SuReal\Documentation\Analisis.txt"/>
    <resource id="LASTOUTPUT" name="last-output" type="data" url="[LASTOUTPUT]"/>
    <resource id="SymtaInputCruiseControl" name="SymTAS CruiseControl" type="data"
        url="[WORKSPACE]\SuReal\SymTASProject\CruiseControl.xml"/>

```

```
146 <resource id="AntScript4CodeGeneration" name="AntSkripte für Codegenerierung"
    type="data" url="[WORKSPACE]\SuReal\build_useful_targets.xml"/>
    <resource id="JamaicaMainClass" name="Jamaica Main Class" type="data"
        url="[WORKSPACE]\SuReal\src\org\surreal\CruiseControl.java"/>
148 <resource id="BrakeControlModel" name="BrakeControl PIM" type="data"
    url="[WORKSPACE]\SuReal\Models\pim\brake_control.uml"/>
    <resource id="CruiseControlModel" name="CruiseControl Model" type="data"
        url="[WORKSPACE]\SuReal\AmeosProjects\CruiseControl\CruiseControl.uml"/>
150
    <resource id="aa01" login="albert" name="Albert Alleskoenner" password="aa"
        status="available" type="person"/>
152 <resource id="mm01" login="maxi" name="Maxi Model" password="mm"
    status="available" type="person"/>
    <resource id="ff01" login="felix" name="Felix Fuchs" password="ff"
        status="available" type="person"/>
154 <resource id="tt01" login="timmy" name="Timmy Tool" password="tt"
    status="available" type="person"/>
    <resource id="ss01" login="susi" name="Susi Sorglos" password="ss"
        status="holiday" type="person"/>
156
</resourceList>
158 </resourceDefinition>
```



## D Vorgehensmodellspezifikationen der Fallstudienbeispiele

### D.1 Erweitertes abstraktes NFE-Vorgehensmodell

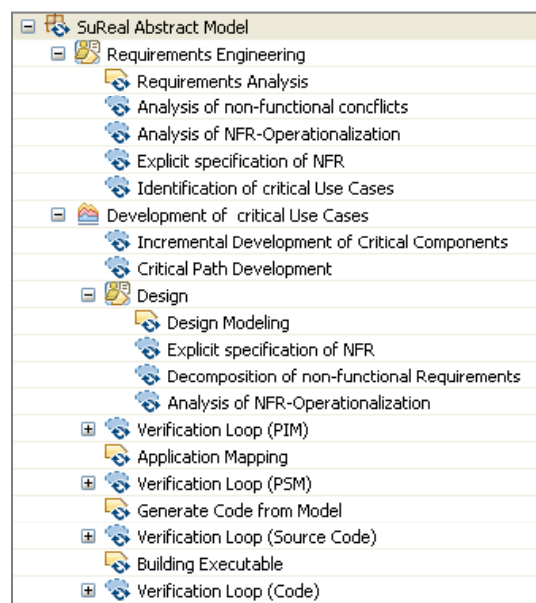


Abbildung D.1: Beispiel eines abstrakten NFE-Vorgehensmodells

Abbildung D.1 zeigt die Ablaufstruktur des abstrakten NFE-Vorgehensmodells der SuReal-Fallstudie. Im Beispiel ist sowohl das NFE-Prozessmuster *Additive Verifikation* als auch die *Transformation der NFE Spezifikation* bereits in einem ersten Iterationsschritt verfeinert worden. Die Additive Verifikation wurde durch die Integration mehrerer Verifikationsschleifen (**Verification Loops**) nach dem entsprechenden Muster umgesetzt. Die *Transformation der NFE Spezifikation* wurde an das in diesem Beispiel eingesetzte Paradigma der modellgetriebenen Entwicklung angepasst und durch die Schritte **Application Mapping**, **Generate Code from Model** und **Building Executables** realisiert.

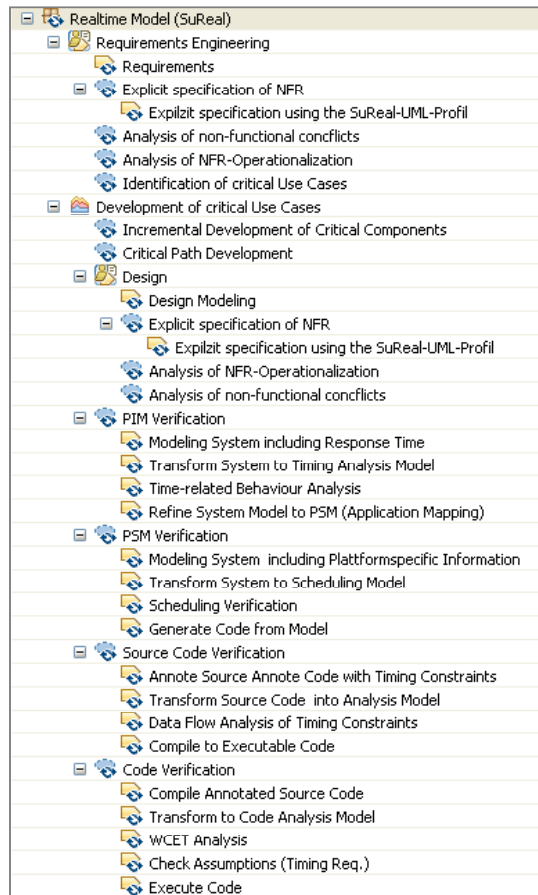


Abbildung D.2: Beispiel eines realzeit-spezifischen NFE-Vorgehensmodells

## D.2 Realzeit-spezifisches Vorgehensmodell

Das konkretisierte NFE-Vorgehensmodell für die Unterstützung von Realzeitanforderungen ist in Abbildung D.2 dargestellt. Als Ausgangsmodell (abstraktes NFE-Vorgehensmodell) diente das Modell der SuReal-Fallstudie. Das in der Abbildung gezeigte realzeit-spezifische NFE-Vorgehensmodell wurde zur Entwicklung der Beispielanwendung (Kapitel 9.4) erfolgreich eingesetzt. Zu sehen ist die Konkretisierung des Musters *Explicit Specification of NFR*. Dieses fordert nun sowohl die Anforderungen als auch die Entwurfsmodelle mit dem SuReal-UML-Profil zu modellieren, welches die explizite Maßangabe von zeitbasierten NFE, wie **Deadlines**, **Execution Budgets**, **Jitter** und **Latency** unterstützt. Ebenfalls konkretisiert wurden die Verifikationsschleifen. Diese verlangen nun realzeitspezifische Verifikationstechniken auf den verschiedenen Ebenen der hier unterstützten modellgetriebenen Entwicklung.

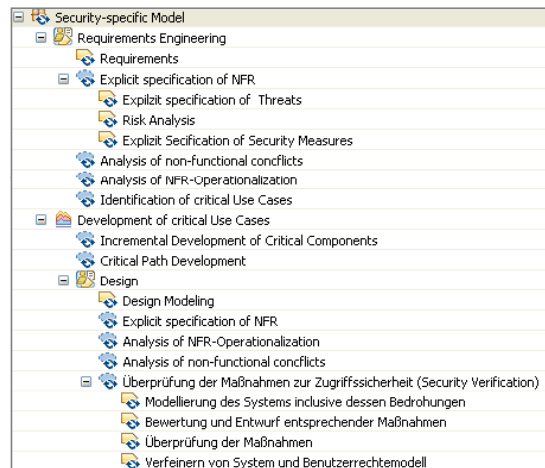


Abbildung D.3: Beispiel eines sicherheits-spezifischen NFE-Vorgehensmodells

## D.3 Sicherheits-spezifisches Vorgehensmodell

Abbildung D.3 zeigt einen Ausschnitt aus der Ablaufstruktur eines sicherheits-spezifischen Vorgehensmodells. Ziel ist die Abbildung der PROKRIS-Konzepte auf das in [Pop05] beschriebene Prozessmodell für sicherheitskritische Systeme. Daher ist das Modell, wie in [Pop05] selbst, auf die frühen Entwicklungsphasen beschränkt. Zu sehen ist die Verfeinerung des Musters *Verifikationsschleife* durch den Prozessbaustein *Überprüfung der Maßnahmen zur Zugriffssicherheit*. Dieser steht bisher nur in deutscher Sprache zur Verfügung (siehe auch Abbildung 5.7) und ist durch den Wissensingenieur noch ins Englische zu übersetzen. Außerdem wurde das Muster *Explizite Spezifikation der NFE* innerhalb der Anforderungsanalyse an die Analyse und Spezifikation der Zugriffssicherheit angepasst. In der Entwurfsphase (Design) wurde dieses Muster dagegen noch nicht konkretisiert.

## D.4 Ausführungsspezifikation des realzeit-spezifischen Vorgehensmodells

Das Ergebnis der Individualisierung des realzeitspezifischen Vorgehensmodells ist in Listing D.1 dargestellt. Die gebundenen konkreten Ressourcen bzw. Ressourcengruppen sind in der Ressourcenspezifikation in Anhang C definiert.

Listing D.1: Beispiel der ausführbaren und individualisierten realzeitspezifischen Workflowspezifikation

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <process-definition xmlns="" name="SuReal Demo – Short Demo">
3   <start-state name="Start">
4     <transition to="Analyse and document requirements"></transition>
5   </start-state>
6
  
```

```

8 <task-node name="Analyse and document requirements">
  <task name="Analyse requirements" description="Analyse the requirements.
    ##surreal\deliveryprocesses\requirements_engineering_84D9D61B.html">
  </task>
10 <task name="Document requirements" description="Dokument the analysed
    requirements.
    ##surreal\deliveryprocesses\requirements_engineering_84D9D61B.html">
  </task>
12 <event type="task-start">
  <action
    class="org.surreal-projekt.eclipse.spms.handler.ResourceAssignmentHandler"
    name="Resource binding">
14 <resourceInterface>
  <role group="Analysis" />
16 <workproduct group="Analysis" />
  <tool group="Analysis" />
18 </resourceInterface>
  </action>
20 </event>
  <transition to="Model PIM with Topcased"></transition>
22 </task-node>

24 <task-node name="Model PIM with Topcased">
  <task name="Model PIM (Topcased)" description="Use UML diagrams like Class,
    State and Sequence Diagrams to describe the design of the
    system.##surreal\deliveryprocesses\design_modeling_91B68291.html">
26 <event type="task-start">
  <action
    class="org.surreal-projekt.eclipse.spms.handler.ResourceAssignmentHandler"
    name="Resource binding">
28 <resourceInterface>
  <role group="PIM.design" />
30 <workproduct group="PIM.design" />
  <tool group="PIM.design" />
32 </resourceInterface>
  </action>
34 </event>
  </task>
  <transition to="Convert to UPPAAL and verify (automatically)"></transition>
36 </task-node>

38 <task-node name="Convert to UPPAAL and verify (automatically)">
40 <task name="Convert to UPPAAL (automatically)" description="The analysis is
    fully automated and must only be initiated by the user. Using the timing
    model created by the task PIM to Timing Analysis Model Transformation the
    UPPAAL tool can check the model. To start this verification the button
    Verify with UPPAAL from the UPPAAL-Tools Plug-in can be used.
    ##surreal\capabilitypatterns\time-related_behaviour_analysis_95B56A88.html">
  <event type="task-start">
42 <action
    class="org.surreal-projekt.eclipse.spms.handler.ResourceAssignmentHandler"
    name="Resource binding">
  <resourceInterface>
44 <role group="PIM.transform" />
  <workproduct group="PIM.transform" />
46 <tool group="PIM.transform" />
  </resourceInterface>
48 </action>
  </event>
50 </task>
  <transition to="Decision: Verification successful?"></transition>
52 </task-node>

```

```

54 <state name="Decision: Verification successful?">
    <transition to="Model PIM with Topcased" name="No. Remodel the
        PIM."></transition>
56 <transition to="Analyze and document requirements" name="No. Remodel the
        requirements."></transition>
    <transition to="Model PSM with Topcased" name="Yes."></transition>
58 </state>

60 <task-node name="Model PSM with Topcased">
    <task name="Model PSM (Topcased)" description="Model
        PSM.##surreal\deliveryprocesses\psm_verification_A7B34420.html">
62 <event type="task-start">
    <action
        class="org.surreal-projekt.eclipse.spms.handler.ResourceAssignmentHandler"
        name="Resource binding">
64 <resourceInterface>
    <role group="PSM.design" />
66 <workproduct group="PSM.design" />
    <tool group="PSM.design" />
68 </resourceInterface>
    </action>
70 </event>
    </task>
72 <transition to="Run scheduling analysis with SymTAS"></transition>
</task-node>
74

<state name="Decision: Result OK?">
76 <transition to="Model PIM with Topcased" name="No. Remodel the
        PIM."></transition>
    <transition to="Analyze and document requirements" name="No. Remodel the
        requirements."></transition>
78 <transition to="Model PSM with Topcased" name="No. Remodel the
        PSM."></transition>
    <transition to="Generate C Code (automatically)" name="Yes."></transition>
80 </state>

82 <task-node name="Generate C Code (automatically)">
    <task name="Generate C code (automatically)" description="Generate C
        code.##surreal\deliveryprocesses\generate_java_code_31E2FE1C.html">
84 <event type="task-start">
    <action
        class="org.surreal-projekt.eclipse.spms.handler.ResourceAssignmentHandler"
        name="Resource binding">
86 <resourceInterface>
    <role group="PSM.generationInC" />
88 <workproduct group="PSM.generationInC" />
    <tool group="PSM.generationInC" />
90 </resourceInterface>
    </action>
92 </event>
    </task>
94 <transition to="Make binarywith GCC (automatically)"></transition>
</task-node>
96

<task-node name="Make binarywith GCC (automatically)">
98 <task name="Compile C code" description="Compile C
        code.##surreal\deliveryprocesses\binary_verification_4E47279B.html">
    <event type="task-start">
100 <action
        class="org.surreal-projekt.eclipse.spms.handler.ResourceAssignmentHandler"
        name="Resource binding">

```

```

102     <resourceInterface>
103     <role group="PDM" />
104     <workproduct group="PDM.compileInC" />
105     <tool group="PDM.compileInC" />
106   </resourceInterface>
107 </action>
108 </event>
109 </task>
110 <transition to="Check WCET assumption with aiT"></transition>
111 </task-node>
112 <task-node name="Check WCET assumption with aiT">
113   <task name="Check WCET assumptions." description="Check WCET
114     assumptions.##surreal\capabilitypatterns\wcet_analysis_CDE737C7.html">
115     <event type="task-start">
116       <action
117         class="org.surreal-projekt.eclipse.spms.handler.ResourceAssignmentHandler"
118         name="Resource binding">
119         <resourceInterface>
120           <role group="PDM" />
121           <workproduct group="PDM.wcet" />
122           <tool group="PDM.wcet" />
123         </resourceInterface>
124       </action>
125     </event>^
126   </task>
127   <transition to="WCET Analysis successful?"></transition>
128 </task-node>
129
130 <task-node name="Deploy to roboter (automatically)">
131   <task name="Deploy to robot." description="Deploy to robot.##surreal\
132     capabilitypatterns\binary2wcet_analysis_model_transformation_B0F51B95.html">
133     <event type="task-start">
134       <action
135         class="org.surreal-projekt.eclipse.spms.handler.ResourceAssignmentHandler"
136         name="Resource binding">
137         <resourceInterface>
138           <role group="PDM.deployToRobot" />
139           <workproduct group="PDM.deployToRobot" />
140           <tool group="PDM.deployToRobot" />
141         </resourceInterface>
142       </action>
143     </event>
144   </task>
145   <transition to="Ende"></transition>##
146 </task-node>
147
148 <state name="WCET Analysis successful?">
149   <transition to="Model PIM with Topcased" name="No. WCET too high."></transition>
150   <transition to="Deploy to roboter (automatically)" name="Yes."></transition>
151 </state>
152
153 <task-node name="Run scheduling analysis with SymTAS">
154   <task name="Run scheduling analysis in SymTA/S." description="Do scheduling
155     analysis.##surreal\capabilitypatterns\
156     binary2wcet_analysis_model_transformation_B0F51B95.html">
157     </task>
158     <transition to="Decision: Result OK?"></transition>
159   </task-node>
160
161 <end-state name="Ende"></end-state>
162 </process-definition>

```

# Literaturverzeichnis

- [ABN<sup>+</sup>09] ALDAZABAL, AITOR, TERRY BAILY, FELIX NANCLARES, ANDREY SADOVYKH, CHRISTIAN HEIN und TOM RITTER: *Automated Model Driven Development Processes*. In: *ECMDA 2008 - Tools & Process Integration Workshop*, Berlin, June 2009.
- [Abs] ABSINT ANGEWANDTE INFORMATIK GMBH: *aiT Worst-Case Execution Time Analyzers*. [www.absint.com/ait](http://www.absint.com/ait).
- [Agi] *Manifesto for Agile Software Development*. <http://agilemanifesto.org/>.
- [Aßm03] ASSMANN, UWE: *Invasive Software Composition*. Springer, 2003.
- [Amb98] AMBLER, SCOTT W.: *Process Patterns: Building Large-Scale Systems Using Object Technology*. Cambridge University Press, 1998.
- [Amb99] AMBLER, SCOTT W.: *More Process Patterns: Delivering Large-Scale Systems Using Object Technology*. Cambridge University Press, 1999.
- [Apa] APACHE: *Apache Ant*. [ant.apache.org](http://ant.apache.org).
- [APRZ03] AIGNER, RONALD, MARTIN POHLACK, SIMONE RÖTTGER und STEFFEN ZSCHALER: *Towards Pervasive Treatment of Non-Functional Properties at Design and Run-Time*. In: *Proc. Intl. Conf. on Software & Systems Engineering and their Applications (ICSSEA'03)*, Paris, France, Dezember 2003.
- [ArE96] AL-RAWAS, AMER und STEVE EASTERBROOK: *Communication Problems in Requirements Engineering: A Field Study*. In: *Proc. of Conf. on Prof. on Awareness in Software Engineering*, 1996.
- [ARNRSG06] AIZENBUD-RESHEF, NETA, BRIAN T. NOLAN, JULIA RUBIN und Yael SHAHAM-GAFNI: *Model traceability*. IBM Systems Journal, 45(3):515–526, 2006.
- [AvEI06] ALMEIDA, JOAO PAULO, PASCAL VAN ECK und MARIA-EUGENIA IACOB: *Requirements Traceability and Transformation Conformance in Model-Driven Development*. Enterprise Distributed Object Computing Conference, IEEE International, 0:355–366, 2006.
- [BA01] BERGMANS, LODEWIJK und MEHMET AKSIT: *Composing crosscutting concerns using composition filters*. Commun. ACM, 44(10):51–57, 2001.
- [Bal00] BALZERT, HELMUT: *Lehrbuch der Softwaretechnik*, Band 1: Software-Entwicklung. Spektrum Akademischer Verlag, 2. Auflage, 2000.
- [Bal08] BALZERT, HELMUT: *Lehrbuch der Softwaretechnik*, Band 2: Softwaremanagement. Spektrum Akademischer Verlag, 2008.
- [Ban04] BANDELOW, DIRK: *Entwicklung einer CQML<sup>+</sup>-Basisbibliothek*. Diplomarbeit, Fakultät Informatik, Technische Universität Dresden, 2004.
- [Bar03] BARTH, MICHAEL N.: *Integration of Simulation Based Performance Assessment in a Software Development Process*. In: DOSCH, WALTER und ROGER Y. LEE (Herausgeber): *Proc. of the ACIS 4th Int. Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'03)*, Lübeck, October 2003. ACIS.
- [BB01] BRETON, ERWAN und JEAN BÉZIVIN: *Model Driven Process Engineering*. Computer Software and Applications Conference, Annual International, 0:225, 2001.
- [BBH<sup>+</sup>03] BREU, RUTH, KLAUS BURGER, MICHAEL HAFNER, JAN JÜRJENS, GERHARD POPP, GUIDO WIMMEL und VOLKMAR LOTZ: *Key Issues of a Formally Based Process Model for Security Engineering*. In: *Proceedings of the 16th International Conference on Software & Systems Engineering and their Applications (ICSSEA03)*, Paris, December 2-4 2003.

## LITERATURVERZEICHNIS

- [BBHP04] BREU, RUTH, KLAUS BURGER, MICHAEL HAFNER und GERHARD POPP: *Towards a Systematic Development of Secure Systems*. In: *WOSIS*, Seiten 1–12, 2004.
- [BCCG07] BENDRAOU, REDA, BENOÎT COMBEMALE, XAVIER CRÉGUT und MARIE-PIERRE GERVAIS: *Definition of an Executable SPEM 2.0*. In: *APSEC*, Seiten 390–397. IEEE Computer Society, 2007.
- [Bec00] BECK, KENT: *Extreme Programming explained*. Addison Wesley, 2000.
- [BJF09] BENDRAOU, REDA, JEAN-MARC JÉZÉQUEL und FRANCK FLEUREY: *Combining Aspect and Model-Driven Engineering Approaches for Software Process Modeling and Execution*. In: WANG, QING, VAHID GAROUSI, RAYMOND J. MADACHY und DIETMAR PFAHL (Herausgeber): *ICSP*, Band 5543 der Reihe *LNCS*, Seiten 148–160. Springer, 2009.
- [BMIS03] BALSAMO, S., A. DI MARCO, P. INVERARDI und M. SIMEONI.: *Software Performance: State of the Art and Perspectives*. Technischer Bericht CS-2003-1 and MIUR Sahara Project TR SAH/001, Dipartimento di Informatica, Università Ca’ Foscari di Venezia, 2003.
- [Boe88] BOEHM, BERRY: *A Spiral Model of Software Development and Enhancement*. IEEE Computer, 21(5):61–72, 1988.
- [Bor04] BORG, ANDREAS: *Contributions to Management and Validation of Non-Functional Requirements*. Licentiate thesis no. 1126, Department of Computer and Information Science, Linköpings Universitet, Sweden, 2004.
- [Bri96] BRINKKEMPER, SJAAK: *Method engineering: engineering of information systems development methods and tools*. Information & Software Technology, 38(4):275–280, 1996.
- [BS04] BALSAMO, SIMONETTA und MARTA SIMEONI: *Integrating Performance Modeling in the Software Development Process*. In: *Radical Innovations of Software and Systems Engineering in the Future*, Band 2941 der Reihe *LNCS*. Springer, 2004.
- [BSGB07] BENDRAOU, REDA, ANDREY SADOVYKH, MARIE-PIERRE GERVAIS und XAVIER BLANC: *Software Process Modeling and Execution: The UML4SPM to WS-BPEL Approach*. In: *EUROMICRO-SEAA*, Seiten 314–321. IEEE Computer Society, 2007.
- [Buh04] BUHL, AXEL: *Grundkurs Software-Projektmanagement*. Carl Hanser Verlag München Wien, 2004.
- [BWHW05] BRAUN, CHRISTIAN, FELIX WORTMANN, MARTIN HAFNER und ROBERT WINTER: *Method construction - a core approach to organizational engineering*. In: HADDAD, HISHAM, LORIE M. LIEBROCK, ANDREA OMICINI und ROGER L. WAINWRIGHT (Herausgeber): *SAC*, Seiten 1295–1299. ACM, 2005.
- [CCCC06] COMBEMALE, BENOÎT, XAVIER CRÉGUT, ALAIN CAPLAIN und BERNARD COULETTE: *Towards a Rigorous Process Modeling with SPEM*. In: MANOLOPOULOS, YANNIS, JOAQUIM FILIPE, PANOS CONSTANTOPOULOS und JOSÉ CORDEIRO (Herausgeber): *ICEIS (3)*, Seiten 530–533, 2006.
- [CD01] CHEESMAN, J. und J. DANIELS: *UML Components: A Simple Process for Specifying Component-Based Software*. Addison Wesley Longman, Inc., 2001.
- [CN92] CHEN, MINDER und RONALD J. NORMAN: *A Framework for Integrated CASE*. IEEE Software, 9(2):18–22, 1992.
- [CNYM00] CHUNG, LAWRENCE, BRIAN A. NIXON, ERIC YU und JOHN MYLOPOULOS: *Non-Functional Requirements In Software Engineering*. Kluwer Academic Publishers, 2000.
- [COM] COMQUAD: *COMponents with QUantitative properties and ADaptivity*. DFG-funded research project, TU Dresden, LMU München, TU Erlangen, 2001-2004.
- [CS06] CHUNG, LAWRENCE und SAM SUPAKKUL: *Representing NFRs and FRs: A Goal-Oriented and Use Case-Driven Approach*. In: DOSCH, W., R. Y. LEE und C. WOO (Herausgeber): *SERA 2004: Revised Selected Papers*, Nummer 3647 in *LNCS*, Seiten 29–41, 2006.



- [DIN97] DIN: *Projektwirtschaft: Projektabwicklung - Begriffe*. DIN 69905, 1997.
- [Dou98] DOUGLASS, BRUCE POWEL: *Real-Time UML: Developing Efficient Objects for Embedded Systems*. Addison Wesley, 1998.
- [DW98] D'SOUZA, D.F. und A.C. WILLS: *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison Wesley Object TEchnology Series. Addison Wesley, 1998.
- [Edg] EDGEWALL SOFTWARE: *Trac -Project Website*. trac.edgewall.org.
- [EPF] *Eclipse Process Framework Project (EPF)*. www.eclipse.org/epf.
- [FCH06] FLETCHER, JESSE und JANE CLELAND-HUANG: *Softgoal Traceability Patterns*. In: *ISSRE '06: Proceedings of the 17th International Symposium on Software Reliability Engineering*, Seiten 363–374, Washington, DC, USA, 2006. IEEE Computer Society.
- [FH93] FEILER, PETER H. und WATTS S. HUMPHREY: *Software Process Development and Enactment: Concepts and Definitions*. In: *ICSP*, Seiten 28–40, 1993.
- [GHJV94] GAMMA, E., R. HELM, R. JOHNSON und J. VLISSIDES: *Design Patterns - Elements of Reuseable Object-Oriented Software*. Addison Wesley Longman, Inc., 1994.
- [Gli05] GLINZ, MARTIN: *Rethinking the Notion of Non-Functional Requirements*. Proc. of the Third World Congress for Software Quality (3WCSQ 2005), Vol. II:55–64, Munich, 2005.
- [Gli07] GLINZ, MARTIN: *On Non-Functional Requirements*. Proceedings of the 15th IEEE International Requirements Engineering Conference, Delhi, India, 2007.
- [GMP<sup>+</sup>03] GNATZ, M., F. MARSCHALL, G. POPP, A. RAUSCH und W. SCHWERIN: *The Living Software Development Process*, 2003.
- [GPA<sup>+</sup>04] GÖBEL, STEFFEN, CHRISTOPH POHL, RONALD AIGNER, MARTIN POHLACK, SIMONE RÖTTGER und STEFFEN ZSCHALER: *The COMQUAD component container architecture*. In: MAGEE, JEFF, CLEMENS SZYPERSKI und JAN BOSCH (Herausgeber): *Proc. 4th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Seiten 315–318, Oslo, Norway, Juni 2004. IEEE.
- [GPR06] GRUHN, VOLKER, DANIEL PIEPER und CARSTEN RÖTTGERS: *MDA - Effektives Software-Engineering mit UML 2 und Eclipse*. Springer Verlag, 2006.
- [GPRZ04] GÖBEL, STEFFEN, CHRISTOPH POHL, SIMONE RÖTTGER und STEFFEN ZSCHALER: *The COMQUAD component model: enabling dynamic selection of implementations by weaving non-functional aspects*. In: *AOSD '04: Proc. of the 3rd International Conference on Aspect-oriented software development*, Seiten 74–82, New York, USA, 2004. ACM Press.
- [Gus08] GUSTAFSSON, BJORN: *OpenUP: The Best of Two Worlds*. Method and Tools, Martining and Associates, Switzerland, Spring, 2008.
- [GY01] GROSS, DANIEL und ERIC YU: *From Non-Functional Requirements to Design through Patterns*. Requirements Engineering, Springer-Verlag, 6:18–36, 2001.
- [HAH05] HENRIKSSON, ANDERS, UWE ASSMANN und JAMES HUNT: *Improving Software Quality in Safety-Critical Applications by Model-Driven Verification*. Electr. Notes Theor. Comput. Sci., 133:101–117, 2005.
- [Har07] HARTWIG, CHRITOPH: *Konzeption und Entwicklung einer allgemeinen workflow-getriebenen Komponentensteuerung am Beispiel JBoss jBPM*. Diplomarbeit, TU Dresden, 2007.
- [Hau06] HAUMER, PETER: *The Eclipse Process Framework Composer. Increasing Development Knowledge with EPFC*. Eclipse Review, 1(2):26–29, Spring 2006.
- [HDHM06] HÖRMANN, KLAUS, LARS DITTMANN, BERND HINDEL und MARKUS MÜLLER: *SPICE in der Praxis. Interpretationshilfe für Anwender und Assessoren*. dpunkt Verlag, Heidelberg, 2006.
- [Hei05] HEINE, CHRISTIAN: *Zielorientierte Requirements Engineering Methoden zur Entwicklung von Agentensystemen*. it - Information Technology, 47(1):20–27, 2005.

## LITERATURVERZEICHNIS

- [Hes00] HESSE, WOLFGANG: *Software-Projektmanagement braucht klare Strukturen - Kritische Anmerkungen zum Rational Unified Process*. In: EBERT, J. und U. FRANK (Herausgeber): *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik. Modellierung 2000*, Seiten 143–150, Koblenz, 2000. Fölbach-Verlag.
- [HH08] HÖHN, REINHARD und STEPHAN HÖPPNER: *Das V-Modell XT*. Springer, 2008.
- [Hof08] HOFFMANN, DIRK W.: *Software-Qualität*. Springer Verlag, Berlin, 2008.
- [Hol02] HOLZ, HARALD: *Process-Based Knowledge Management Support for Software Engineering*. Doktorarbeit, Universität Kaiserslautern, 2002.
- [HRW09] HEIN, C., T. RITTER und M. WAGNER: *Model-Driven Tool Integration with ModelBus*. In: *1st Workshop Future Trends of Model-Driven Development (FTMDD)*, Milan, Italy, 2009.
- [HS03] HENDERSON-SELLERS, BRIAN: *Method Engineering for OO System Development*. Communications of the ACM, 46(10), October 2003.
- [HSGP05] HENDERSON-SELLERS, B. und C. GONZALES-PERES: *A comparison of four process meta-models and the creation of a new generic standard*. Information and Software Technology, 47:49–65, 2005.
- [Hun04] HUNT, JAMES J. (Herausgeber): *The HIDOORS Methodology: Using Java in Realtime and Embedded Systems*. aicas GmbH, 2004.
- [HZP<sup>+</sup>07] HÄRTIG, HERMANN, STEFFEN ZSCHALER, MARTIN POHLACK, RONALD AIGNER, STEFFEN GÖBEL, CHRISTOPH POHL und SIMONE RÖTTGER: *Enforceable Component-Based Realtime Contracts – Supporting Realtime Properties from Software Development to Execution*. Springer Real-Time Systems Journal, 35(1), Januar 2007.
- [IBM] IBM: *Rational Method Composer*. <http://www-01.ibm.com/software/awdtools/rmc/>.
- [IBM03] IBM: *Business Process Execution Language*, May 2003.
- [IEE90] IEEE - INSTITUTE OF ELECTRIC AND ELECTRONIC ENGINEERING: *IEEE Standard Glossary of Software Engineering Terminology*. Technischer Bericht IEEE Std 610.12-1990, IEEE, New York, 1990.
- [INC] INCOSE (INTERNATIONAL COUNCIL OF SYSTEMS ENGINEERING): *Requirements Management Tools Survey*. [www.paper-review.com/tools/rms/read.php](http://www.paper-review.com/tools/rms/read.php).
- [ISO] ISO/IEC: *Information technology Process assessment*. ISO/IEC 15504.
- [ISO91] ISO/IEC: *Information Technology, Software Product Evaluation, Quality, Characteristics and Guidelines for their Use*. ISO IEC 9126, 1991.
- [ISO07] ISO/IEC: *Software Engineering-Metamodel for Development Methodologies*. ISO/IEC 24744, 2007.
- [jBP] *jBPM - Java Business Process Management Project by JBoss*. [www.jboss.org/jbossjbpm](http://www.jboss.org/jbossjbpm).
- [Jen08] JENNY, BRUNO: *Projektmanagement*. vdf Hochschulverlag, Zürich, 2008.
- [Joh09] JOHANNSEN, FLORIAN: *Konzeption und Evaluation eines Ansatzes zur Methodenintegration im Qualitätsmanagement*. In: EYMANN, TORSTEN (Herausgeber): *Tagungsband zum Doctoral Consortium der WI 2009*, Band 40 der Reihe *Bayreuther Arbeitspapiere zur Wirtschaftsinformatik*, Seiten 85–94. Lehrst. für Wirtschaftsinformatik, Univ. Bayreuth, 2009.
- [jPD] *JBoss jBPM jPDL 3.2 - jBPM jPDL User Guide*. <http://docs.jboss.org/jbpm/v3/userguide/>.
- [Jür05] JÜRJENS, JAN: *Secure Systems Development with UML*. Springer, 2005.
- [KBS09] KBST: *V-Modell XT - Dokumentation, Version 1.3*. [www.kbst.bund.de](http://www.kbst.bund.de), 2006, überarbeitete Version von Januar 2009.

- [KL98] KOBIALKA, HANS-ULRICH und CLAUS LEWERENTZ: *User Interfaces Supporting the Software Process*. Seiten 60–74, 1998.
- [KLL09] KO, RYAN K. L., STEPHEN S. G. LEE und E. W. LEE: *Business Process Management (BPM) Standards: A Survey*. Business Process Management Journal, to be published in 15(3), 2009.
- [Kne07] KNEUPER, RALF: *CMMI - Verbesserung von Software- und Systemen mit Capability Maturity Model Integration (CMMI-DEV)*. dpunkt.verlag Heidelberg, 3. Auflage, 2007.
- [KNS92] KELLER, G., M. NÜTTGENS und A.-W. SCHEER: *Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK)*. Veröffentlichungen des Instituts für Wirtschaftsinformatik, 89, 1992.
- [Kru03] KRUCHTEN, PHILIPP: *The Rational Unified Process: An Introduction*. Add. Wesley, 2003.
- [KS98] KOTONYA, GERALD und IAN SOMMERVILLE: *Requirements Engineering - Processes and Techniques*. John Wiley & Sons., May 1998.
- [Kuh08] KUHRMANN, MARCO: *Automatisches, werkzeugspezifisches Tailoring für das V-Modell®XT*. In: *Proc. des 15. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI)*, Seiten 84–93. Shaker Verlag, April 2008.
- [KWB03] KLEPPE, A., J. WARMER und W. BAST: *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison Wesley Professional, April 2003.
- [Lap05] LAPOUCHNIAN, A.: *Goal-Oriented Requirements Engineering: An Overview of the Current Research*. Technischer Bericht, University of Toronto, 2005.
- [Lie06] LIEBSCHNER, MARKUS: *Definition eines erweiterbaren und flexiblen Workflow Management System - Modellierung und Architektur*. Diplomarbeit, TU Dresden, 2006.
- [LL07] LUDEWIG, J. und H. LICHTER: *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. dpunkt.verlag Heidelberg, 2007.
- [LMS03] LAVAGNO, LUCIANO, GRANT MARTIN und BRAN SELIC (Herausgeber): *UML for Real: Design of Embedded Real-Time Systems*. Kluwer Academic Publishers, 2003.
- [LX99] LEE, JONATHAN und NIEN-LIN XUE: *Analyzing User Requirements by Use Cases: A Goal-Driven Approach*. IEEE Software, 16(4):92–101, 1999.
- [Mar05] MARGARIA, TIZIANA: *Web services-based tool-integration in the ETI platform*. Software and System Modeling, 4(2):141–156, 2005.
- [MB01] MALAN, RUTH und DANA BREDEMEYER: *Defining Non-functional Requirements*. Brede-meyer Consulting, White Paper. <http://www.bredemeyer.com/papers.htm>, 2001.
- [MHDZ07] MÜLLER, MARKUS, KLAUS HÖRMANN, LARS DITTMANN und JÖRG ZIMMER: *Automotive SPICE in der Praxis. Interpretationshilfe für Anwender und Assessoren*. dpunkt Verlag, Heidelberg, 2007.
- [Mül07] MÜLLER, SASCHA: *Modellbasierte IT-Unterstützung von wissensintensiven Prozesse - Dargestellt am Beispiel medizinischer Forschungsprozesse*. Doktorarbeit, Universität Erlangen, 2007.
- [MNS05] MARGARIA, TIZIANA, RALF NAGEL und BERNHARD STEFFEN: *Remote Integration and Coordination of Verification Tools in JETI*. In: *ECBS '05: Proc. of the 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, Seiten 431–436, Washington, USA, 2005. IEEE Computer Society.
- [Mor07] MOREIRA, A. (Herausgeber): *Early Aspects: Current Challenges and Future Directions, 10th International Workshop, Revised Selected Papers Series*, Lecture Notes in Computer Science, Vancouver, Canada, March, 13 2007. Springer.
- [Mut07] MUTHMANN, KLEMENS: *Support of a Multistep Workflow Definition based on jBPM and Eclipse*. Großer Beleg, TU Dresden, Dezember 2007.

## LITERATURVERZEICHNIS

- [Neu02] NEUMANN, OLAF: *Integration von Werkzeugen in heterogene, prozessgesteuerte Software-Entwicklungsumgebungen - Tool Integration Concept*. Doktorarbeit, Universität-Gesamthochschule Paderborn, 2002.
- [NR08] NIKNAFS, ALI und RAMAN RAMSIN: *Computer-Aided Method Engineering: An Analysis of Existing Environments*. In: BELLAHSENE, ZOHRA und MICHEL LÉONARD (Herausgeber): *CAiSE*, Band 5074 der Reihe *LNCS*, Seiten 525–540. Springer, 2008.
- [oAW] *openArchitectureWare (oAW)*. [www.openarchitectureware.org](http://www.openarchitectureware.org).
- [Obj03] OBJECT MANAGEMENT GROUP: *MDA Guide Version 1.0.1*. OMG Document, June 2003. <http://www.omg.org/cgi-bin/doc?omg/03-06-01>.
- [Obj05] OBJECT MANAGEMENT GROUP: *UML Profile for Schedulability, Performance and Time*, Januar 2005. <http://www.omg.org/cgi-bin/doc?formal/2005-01-02>.
- [Obj06] OBJECT MANAGEMENT GROUP: *Meta Object Facility (MOF) Core Specification*. OMG Document, Januar 2006. <http://www.omg.org/docs/formal/06-01-01.pdf>.
- [Obj08a] OBJECT MANAGEMENT GROUP: *SPEM 2.0 Software and Systems Process Engineering Metamodel Specification*. OMG Document, April 2008. <http://www.omg.org/cgi-bin/doc?ptc/2008-04-01>.
- [Obj08b] OBJECT MANAGEMENT GROUP: *UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms*, April 2008. <http://www.omg.org/docs/formal/08-04-05.pdf>.
- [Obj09] OBJECT MANAGEMENT GROUP: *Business Process Modeling Notation (BPMN)*, March 2009. <http://www.omg.org/docs/formal/09-01-03.pdf>.
- [Ost87] OSTERWEIL, LEON J.: *Software processes are software too*. In: *ICSE '87: Proceedings of the 9th international conference on Software Engineering*, Seiten 2–13, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.
- [Ost05] OSTERWEIL, LEON J.: *Unifying Microprocess and Macroprocess Research, Invited Keynote Address*. In: MINGSHU LI, BARRY BOEHM und LEON J. OSTERWEIL (Herausgeber): *International Software Process Workshop (SPW2005)*, Band 3840 der Reihe *LNCS*, Beijing, China, 2005. Springer-Verlag.
- [PMI04] PMI (Herausgeber): *A Guide to the Project Management Body of Knowledge: PMBOK Guide*. PMI, 2004.
- [Poh07] POHL, KLAUS: *Requirements Engineering*. dpunkt.verlag Heidelberg, 2007.
- [Pop05] POPP, GERHARD: *Methode zur Integration von Sicherheitsanforderungen in die Entwicklung zugriffssicherer Systeme*. Doktorarbeit, Technische Universität München, 2005.
- [PP03] POPPENDIECK, MARY und TOM POPPENDIECK: *Lean Software Development - An Agile Toolkit*. Addison Wesley, 2003.
- [PR09] PALLUCH, JENS und CHRISTINA ROMCEA: *Metamodell-basierte Modellierung eines Software-Entwicklungsprozesses in der Automobilindustrie: SPEM und ISO/IEC 24744 im Vergleich*. In: *Software and Systems Engineering Essentials (SEE)*, 2009.
- [Pri99] PRISS, UTA: *Description Logic and Faceted Knowledge Representation*. In: *Proceedings of the International Workshop on Description Logics (DL99)*, Band 22 der Reihe *CEUR Workshop Proceedings*, 1999.
- [Pri08] PRISS, UTA: *Facet-like Structures in Computer Science*. *Axiomathes*, 18:243–255, 2008.
- [Ran67] RANGANATHAN, S. R.: *Prolegomena to Library Classification*. Asian Publishing House, Bombay, India, 1967.
- [Rau01] RAUSCH, ANDREAS: *Componentware: Methodik des evolutionären Architekturentwurfs*. Doktorarbeit, Technische Universität München, November 2001.
- [R.C05] R.CHITCHYAN, A.RASHID, P.SAWYER: *Comparing Requirements Engineering Approaches for Handling Crosscutting Concerns*. In: *Workshop on Requirements Engineering (held with CAiSE)*, Porto, Portugal, June 12-14 2005.

- [RFKR00] RUPPRECHT, CHRISTIAN, MARTIN FÜNFFINGER, HOLGER KNUBLAUCH und THOMAS ROSE: *Capture and Dissemination of Experience about the Construction of Engineering Processes*. In: WANGLER, BENKT und LARS BERGMAN (Herausgeber): *CAiSE*, Band 1789 der Reihe *Lecture Notes in Computer Science*, Seiten 294–308. Springer, 2000.
- [Rie03] RIEHLE, DIRK: *The Perfection of Informality: Tools, Templates, and Patterns*. Cutter IT Journal, 16(9):22–26, September 2003.
- [Roy70] ROYCE, WINSTON: *Managing the Development in Large Software Systems*. Proceedings of IEEE WESCOM, 1970.
- [Roy98] ROYCE, WALKER: *Software Project Management - A Unified Framework*. Add. Wesley, 1998.
- [RR01] RALYTÉ, JOLITA und COLETTE ROLLAND: *An Assembly Process Model for Method Engineering*. In: DITTRICH, KLAUS R., ANDREAS GEPPERT und MOIRA C. NORRIE (Herausgeber): *CAiSE*, Band 2068 der Reihe *LNCS*, Seiten 267–283. Springer, 2001.
- [RSS09] RÖTTGER, SIMONE, MIRKO SEIFERT und KATJA SIEGEMUND: *SuReal - Anwendungs- und Entwicklungsdokumentation*. Technischer Bericht, TU Dresden, 2009. zur Veröffentlichung geplant.
- [RZ95] RIEHLE, DIRK und HEINZ ZÜLLIGHOVEN: *A Pattern Language for Tool Construction and Integration Based on the Tools&Materials Metaphor*. In: *Pattern Languages of Program Design*, Seiten 9–42. Addison-Wesley, 1995.
- [RZ03] RÖTTGER, SIMONE und STEFFEN ZSCHALER: *CQML<sup>+</sup>: Enhancements to CQML*. In: *Proc. 1st Int. Workshop on Quality of Service in Component-Based Software Engineering, Toulouse, France*, Seiten 43–56. Cépaduès-Éditions, 2003.
- [RZ04a] RÖTTGER, SIMONE und STEFFEN ZSCHALER: *Model-Driven Development for Non-functional Properties: Refinement through Model Transformation*. In: *Proceedings of the UML Conference, Lisbon, Portugal*, Band 3273 der Reihe *LNCS*. Springer, 2004.
- [RZ04b] RÖTTGER, SIMONE und STEFFEN ZSCHALER: *A Software Development Process Supporting Non-functional Properties*. In: *Proceedings of the IASTED International Conference on Software Engineering (IASTED SE 2004)*, Innsbruck, Australia, 2004. ACTA Press.
- [RZ07] RÖTTGER, SIMONE und STEFFEN ZSCHALER: *Tool Support for Refinement of Non-functional Specifications*. Software and Systems Modeling journal (SoSyM), 6(2), Juni 2007.
- [Sca02] SCACCHI, WALT: *Process Models in Software Engineering*, Kapitel Process Models in Software Engineering, Seiten 993–1005. John Wiley and Sons, Inc, New York, 2. Auflage, 2002.
- [Sch96] SCHEER, AUGUST-WILHELM: *ARIS-House of Business Engineering*. Forschungsberichte des Instituts für Wirtschaftsinformatik, 133, 1996.
- [Sch04] SCHWABER, KEN: *Agile Project Management with Scrum*. Microsoft Press, 2004.
- [Sch07] SCHMIDT, MATTHIAS: *Modellierung und Realisierung einer Workflowsteuerung für Quality Driven Design*. Großer Beleg, TU Dresden, November 2007.
- [Sch09] SCHMITZ, UWE: *Erstellung und Bereitsstellung von Prozessbeschreibungen, die Mitarbeiter nützlich finden*. In: *Software and Systems Engineering Essentials (SEE)*, 2009.
- [SCLJO07] SIMIDCHIEVA, BORISLAVA I., LORI A. CLARKE und LEON J. LEON J. OSTERWEIL: *Representing Process Variation with a Process Family*. In: WANG, QING, DIETMAR PFAHL und DAVID M. RAFFO (Herausgeber): *ICSP*, Band 4470 der Reihe *LNCS*, Seiten 109–120. Springer, 2007.
- [SE03] SKENE, JAMES und WOLFGANG EMMERICH: *A Model-Driven Approach to Non-Functional Analysis of Software Architectures*. Intern. Conf. on Automated Software Engineering, 2003.
- [Sec] *SecurityPattern.org*. [www.securitypatterns.org](http://www.securitypatterns.org). last visited in November 2008.

## LITERATURVERZEICHNIS

- [SFBH<sup>+</sup>05] SCHUMACHER, MARKUS, EDUARDO FERNANDEZ-BUGLIONI, DUANE HYBERTSON, FRANK BUSCHMANN und PETER SOMMERLAD: *Security Patterns: Integrating Security and Systems Engineering*. Wiley and Sons, 2005.
- [SG02] SMITH, CONNIE U. und LLOYD G. WILLIAMS: *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Object-Technology Series. Add.-Wesley, 2002.
- [Sha02] SHAW, MARY: *What makes good research in software engineering?* STTT, 4(1):1–7, 2002.
- [SJP<sup>+</sup>07] SANCHO, PERE P., CARLOS JUIZ, RAMON PUIGJANER, LAWRENCE CHUNG und NARY SUBRAMANIAN: *An Approach to Ontology-aided Performance Engineering through NFR Framework*. In: *WOSP '07: Proc. of the 6th International Workshop on Software and Performance*, Seiten 125–128, New York, NY, USA, 2007. ACM.
- [SL07] SADANA, VISHAL und XIAOQING FRANK LIU: *Analysis of Conflicts among Non-Functional Requirements Using Integrated Analysis of Functional and Non-Functional Requirements*. Computer Software and Applications Conference, Annual International, 1:215–218, 2007.
- [Smi90] SMITH, CONNIE U.: *Performance Engineering of Software System*. Addison Wesley, 1990.
- [SO01] SINDRE, GUTTORM und ANDREAS L. OPDAHL: *Capturing Security Requirements through Misuse Cases*. In: *Proceedings of the Norsk Informatikkonferanse (NIK)*, 2001.
- [Sof] SOFTWARE ENGINEERING INSTITUTE: *Capability Maturity Model Integration (CMMI)*. [www.sei.cmu.edu/cmml](http://www.sei.cmu.edu/cmml).
- [Som07] SOMMERVILLE, JAN: *Software Engineering*. Addison-Wesley, 8th Auflage, 2007.
- [SPGM05] SABETTA, ANTONINO, DORINA C. PETRIU, VINCENZO GRASSI und RAFFAELA MIRAN-DOLA: *Abstraction-raising Transformation for Generating Analysis Models*. In: *Proc. of MoDELS'2005 Satellite Events*, Nummer 3844 in LNCS, Seiten 217–226. Springer, 2005.
- [SuR] *SuReal: Sicherheitsgarantien unter Realzeitanforderungen*. [www.sureal-projekt.org](http://www.sureal-projekt.org).
- [SV05] STAHL, THOMAS und MARKUS VÖLTER: *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. dpunkt.verlag Heidelberg, 2005.
- [SW03] SMITH, CONNIE U. und LLOYD G. WILLIAMS: *UML for Real: Design of Embedded Real-Time Systems*, Kapitel Software Performance Engineering. Kluwer, 2003.
- [Sym] SYMTAVISION: *SymTA/S*. [www.symtavision.com](http://www.symtavision.com).
- [Szy02] SZYPERSKI, CLEMENS: *Component Software: Beyond Object-Oriented Programming*. Component Software Series. Addison-Wesley, 2. Auflage, 2002.
- [TC99] TRAN, QUAN und LAWRENCE CHUNG: *NFR-Assistant: Tool Support for Achieving Quality*. In: *ASSET '99: Proceedings of the 1999 IEEE Symposium on Application - Specific Systems and Software Engineering and Technology*, Seite 284, Washington, DC, USA, 1999. IEEE Computer Society.
- [Tid05] TIDWELL, JENIFER: *Designing Interfaces: Patterns for Effective Interaction Design*. O'Reilly Media, Inc., 2005.
- [TN92] THOMAS, IAN und BRIAN A. NEJMEH: *Definitions of Tool Integration for Environments*. IEEE Softw., 9(2):29–35, 1992.
- [TT05] TONU, SUBRINA ANJUM und LADAN TAHVILDARI: *Towards a Framework to Incorporate NFRs into UML Models*. In: *Proc. of 1st International Workshop on Reverse Engineering to Requirements (RETR'05)*, Seiten 13–18, Carnegie Mellon University, November 2005.
- [UPP] UPPAAL. <http://www.uppaal.com/>. last visited June 2009.
- [vdAtH05] AALST, W.M.P. VAN DER und A.H.M. TER HOFSTEDE: *YAWL: Yet Another Workflow Language*. Information Systems, 30(4):245–275, 2005.
- [Wal01] WALLMÜLLER, ERNEST: *Software-Qualitätsmanagement in der Praxis*. Hanser Verlag, 2001.

- [Wor95] WORKFLOW MANAGEMENT COALITION (WFMC): *The Workflow Reference Model*, 1995.
- [Wor08] WORKFLOW MANAGEMENT COALITION: *XML Process Definition Language Specification 2.1*, October 2008.
- [yAS04] AAGEDAL, JAN ØYVIND und IDA SOLHEIM: *New Roles in Model-Driven Development*. In: *Second European Workshop on Model Driven Architecture (MDA)-EWMDA-2*, Canterbury, England, Sept 2004.
- [YdPLM04] YU, YIJUN, JULIO CESAR SAMPAIO DO PRADO LEITE und JOHN MYLOPOULOS: *From Goals to Aspects: Discovering Aspects from Requirements Goal Models*. In: *RE '04: Proc of the Requirements Engineering Conference*, Seiten 38–47, Washington, USA, 2004. IEEE.
- [YLW06] YUAN, FENG, MINGSHU LI und ZHIGANG WAN: *SPEM2XPDL-Towards SPEM Model Enactment*. In: *International Conference on Software Engineering Research and Practice (SERP)*, Las Vegas, Nevada, USA, 06.
- [YT97] YEN, JOHN und W. AMOS TIAO: *A Systematic Tradeoff Analysis for Conflicting Imprecise Requirements*. In: *Proc. of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, Seiten 5–8, 1997.
- [ZGK04] ZUSER, WOLFGANG, THOMAS GRECHING und MONIKA KÖHLE: *Software Engineering mit UML und dem Unified Process*. Pearson Studium, 2. Auflage, 2004.
- [ZHG05] ZUSER, WOLFGANG, STEFAN HEIL und THOMAS GRECHENIG: *Software Quality Development and Assurance in RUP, MSF and XP: A Comparative Study*. In: *3-WoSQ: Proc. of the 3rd Workshop on Software Quality*, Seiten 1–6, New York, USA, 2005. ACM.
- [Zül98] ZÜLLIGHOVEN, HEINZ: *Das objektorientierte Konstruktionshandbuch nach dem Werkzeug und Material-Ansatz*. dpunkt Verlag, 1998.
- [ZR04] ZSCHALER, STEFFEN und SIMONE RÖTTGER: *Types of Quality of Service Contracts for Component-Based Systems*. In: *Proc. IASTED Int'l Conf. on Software Engineering (IASTED SE 2004)*. ACTA Press, 2004.
- [Zsc] ZSCHALER, STEFFEN: *Bibliography on Non-functional Properties and Requirements*. [www.steffen-zschaler.de](http://www.steffen-zschaler.de).
- [Zsc07] ZSCHALER, STEFFEN: *A Semantic Framework for Non-functional Specifications of Component-Based Systems*. Dissertation, TU Dresden, Germany, April 2007.
- [Zsc09] ZSCHALER, STEFFEN: *Formal Specification of Non-functional Properties of Component-Based Software Systems: A Semantic Framework and Some Applications Thereof*. Software and Systems Modelling (SoSyM), 2009. To appear.